Manuale e Ambiti di Utilizzo per ntJobApps e ntJobsOS
Versione 20251227

Sommario

1. Introduzione e Scopo

Una ntJobApp è un'applicazione progettata per funzionare in **modalità batch**, comunicando con l'utente tramite un file di ritorno, quindi una applicazione asincrona.

Lo scopo di una ntJobApp è triplice:

1. Leggere un file .ini passato come parametro (di solito di nome ntjobs.ini)
2. Eseguire uno o più comandi richiesti dal chiamante.
3. Restituire un file .ini di risultato delle elaborazioni. Il file di output .ini verrà rinominato con estensione .end.

4. Le ntjobsapp sono costruite per essere orchestrate da ntjobsos, ma possono funzionare anche autonomamente.

2. Logica di Funzionamento per l'Esecutore

L'esecuzione di una ntJobApp segue una logica sequenziale standard:

1. **Creazione Istanza:** Viene creata l'istanza di ncJobApp nella variabile globale jData.
2. **Avvio (Start()):** Viene eseguito jData.Start(), che legge il file .ini passato come parametro (o lo crea) e lo salva in self.dictJobs.
    - Se sResult (il risultato dell'operazione) è diverso da "", l'applicazione esce con errorlevel **1**.
3. **Esecuzione (Run()):** Viene eseguito self.Run(), che esegue ogni singolo job definito in self.dictJobs in sequenza.
4. **Finalizzazione (End()):** Viene eseguito self.End(sResult).

3. Istruzioni per l'Esecuzione

Per eseguire una ntJobApp, è necessario avviare l'applicazione passando come **primo parametro** il percorso completo del file di configurazione .ini.

3.1. Il File di Input (.ini)

Il file .ini deve contenere almeno la sezione [CONFIG] e una o più sezioni di *job*.

Esempio di file ntjobsapp.ini letto all'avvio:

Ini, TOML

```
[CONFIG]

TYPE=NTJOBS.APP.1.0

EXIT=TRUE      ; Se "TRUE", l'applicazione esce in caso di errore di un job.

LOG="FILE.LOG"  ; Nome facoltativo del file di Log.

NAME=ID_APPLICAZIONE

[JOB1_ID]

ACTION=NOME_AZIONE

FILE.ID1=PathFile1

PARAM.ID1=Valore
```

Validazioni cruciali in Start():

- **Sezione CONFIG:** Deve essere presente nel dizionario letto in self.dictJobs, altrimenti l'applicazione esce con l'errore "Sezione CONFIG non trovata".
- **Chiave COMMAND:** In ogni sezione job (eccetto CONFIG), deve esserci la chiave COMMAND (o ACTION nell'esempio), altrimenti l'errore viene accodato a sResult.
- **Verifica file:** Se ci sono chiavi che cominciano per FILE., l'applicazione verifica che il file esista. Se un file non esiste, l'errore viene accodato a sResult, ma l'esecuzione **non** viene interrotta in questa fase.

3.2. Il File di Output (.end)

Al termine dell'esecuzione, il metodo End(sResult) salva il contenuto di self.dictJobs come file .ini nel percorso self.sJobEnd. self.sJobEnd è il nome di self.sJobIni con l'estensione cambiata in .end.

Il file .end conterrà i risultati di ogni job, inclusi:

- RETURN.TYPE (es. E=Errore, ""=OK).
- RETURN.VALUE (Messaggio di ritorno facoltativo).
- TS.START e TS.END (Timestamp di inizio e fine).
- Chiavi con prefisso FILE.RETURN. (riferimenti a file generati dal job da ritornare all'utente).

3.3. Codici di Risultato (Errorlevel)

Al termine dell'esecuzione, la ntJobApp restituisce un codice di risultato (errorlevel) che indica lo stato finale:

Codice	Descrizione	Significato
0	Tutto a posto	L'applicazione è terminata correttamente, tutti i job sono stati eseguiti.
1	Non è riuscito ad arrivare all'esecuzione dei job	Errore critico in fase di avvio (Start()), il file .ini di fine elaborazione non viene generato.
2	Uno o più job è terminato con un errore	Uno o più job all'interno del file .ini sono terminati con un errore.

4. Ambiti di Utilizzo e Vantaggi vs. Automazione Web (Clic e Attese)

L'architettura ntJobApp è specificamente progettata per processi di automazione **batch** e offre significativi vantaggi rispetto a soluzioni che richiedono interazione grafica (come clic e attese su un sito web).

Ambiti Tipici di Utilizzo di ntJobApp	**Vantaggi Chiave Rispetto all'Automazione Web (GUI)**
Processi Asincroni: Elaborazioni complesse o massive, come la generazione di report notturni, che non devono bloccare l'utente.	**Efficienza e Velocità:** Essendo un processo di *backend*, non ha overhead grafici, non richiede l'attesa di rendering della pagina o l'interazione umana, riducendo drasticamente i tempi morti.
Integrazione Dati: Spostamento, conversione o elaborazione massiva di file (es. da FTP a un database), processi che lavorano "sotto il cofano".	**Robustezza e Affidabilità:** Non è soggetta a errori dovuti a cambiamenti nell'interfaccia utente (come un bottone che cambia ID) o problemi di timeout della rete dovuti al caricamento di una pagina.
Orchestrazione Strutturata: Chiamare diverse applicazioni sequenzialmente, dove i risultati e i file di ritorno di un'azione vengono gestiti come input per la successiva.	**Tracciabilità Standardizzata:** Il formato .ini in ingresso e l'output .end garantiscono un formato di status uniforme e leggibile dal chiamante, inclusi i file di ritorno generati.
Schedulazione: Esecuzione di task automatici a orari predefiniti (come Cron o Task Scheduler).	**Facilità di Automazione:** L'esecuzione da linea di comando con file di parametri la rende ideale per qualsiasi sistema di *job scheduler*, richiedendo solo il percorso del file .ini.

5. Particolarità di una NTJOBSAPP e di NTJOBSOS

Una NTJOBSAPP deve essere semplice perché portatile e rivolta all’automazione e asincronicità nella comunicazione tra utente e macchina.

Per alcuni aspetti il paradigma è lo stesso di altri **fenomeni tecnologici:**

5.1. Similitudini con Docker

- La composizione di una immagine docker non è fatta con un sequenza di click e command line di install, ma con un file yaml che dicono al docker composer in sequenza cosa deve fare

5.2. Similitudini con l'architettura a MicroServizi

Moderno fenomeno tecnologico con cui sono "scomposte" e "riutilizzabili" le applicazioni scritte a "moduli autonomi". Una ntJobsApp è composta da moduli "autonomamente" richiamabili dall'utente, i "command" passando alla ntJobsApp i parametri di esecuzione. Le stesse ntJobsapp sono richiamate da un "orchestratore superiore", ntjobsos che invia a una o più ntjobsapp i parametri di esecuzione da parte di utenti "riconosciuti" ed "autorizzati" ad utilizzarla sulla piattaforma di esecuzione dove ntjobsos funziona.

5.3. Sostenibilità di una applicazione ntjobsapp, della piattaforma di erogazione

Una ntjobsapp può essere vista come "prodotto" secondo un certo paradigma di esecuzione e produzione e ntjobsos lo store (negozio), tramite il quale vengono utilizzate le applicazioni.

Il termine alla base del paradigma di questa coppia è la sostenibilità e vari compromessi per mantenere questo principio di fondo. Ecco i compromessi:

- Non c'è interfaccia WEB (o almeno non nelle ntjobsapp) . L'interfaccia con l'utente finale è con canali "semplici" e l'esecuzione è asincrona.
- Dal punto di vista del rapporto tra produttore di una ntjobsapp e il suo utilizzatore il patto di produzione è questo: Mi dici cosa vuoi e io ti faccio la app chiavi in mano (e solo di tipo batch via file Cloud mail).
- L'end user può essere lo stesso produttore e le ntjobsapp prodotte servono solo per scopi personali. In tal caso il produttore possiede anche lo store di distribuzione con una propria installazione di ntjobsos, ma anche no, perché le ntjobsapp devono poter essere utilizzabili anche autonomamente come normali applicazioni command line.

5.4. Struttura informatica di una NTJOBSAPP

- Le ntjobsapp e ntjobsos devono essere facilmente spostabili e poco costose da mantenere nell'infrastruttura dove vengono eseguite. Deve esserci il linguaggio di esecuzione previsto (PythonX64 portable e le librerie previste).
- Nulla vieta che le ntjobsapp e ntjobsos mediante il prompt di generazione, siano "generate" anche in altri linguaggi oltre Python (Java, Vba, Rust, ecc).
- Data una soluzione per un problema progettata per risolvere il problema dell'utente finale, il costo dello store e di produzione dell'applicazione deve essere il più basso possibile.

- L’infrastruttura sottostante a ntjobsos e del framework ntjobsapp deve essere meno semplici. Non vengono utilizzati database sottostanti ne webserver. I dati di configurazione di partenza sono gestiti tramite file .csv e .ini
- ntjobsos si appoggia al sistema operativo sottostante mediante il layer python (o altri linguaggi installati se viene generato anche per quel linguaggio). Le ntjobsapp possono essere scritte anche in linguaggi diversi, purché integrino la classe di “interfaccia” (ncJobsApp) se “prevista” per quel linguaggio e devono essere costruite come una collezione di “comandi” eseguibili in sequenza mediante il file .ini passato come parametro, e i parametri di lancio dei “commands” previsti nell’applicazione.
- L’interfaccia da includere in una ntjobsapp è composta da:
    - aiSys.py o nlSys.py: Libreria di funzioni e classi “generiche”
    - acJobsApp.py o ncJobsApp.py: Classe di interfaccia e gestione della ntJobsApp
    - Funzione Start di inizializzazione jData come istanza di acJobsOS/ncJobsOS e cbCommands() callback per eseguire i vari comandi
- Le ntjobsapp possono eventualmente utilizzare le classi del sistema ntJobsOS, quando queste sono testate e verificate. Da valutarsi caso per caso. Possono anche essere NON generate da AI ma scritte manualmente, ma devono utilizzare il paradigma di funzionamento delle ntjobsapp ed includere le classi di interfaccia ncJobsApp o acJobsApp e governate dall’istanza jData di queste classi che hanno scopi e struttura uguale.

5.5. Struttura informatica di ntJobsOS

- ntJobsOS.py / niJobsAI.py è l’applicativo di esecuzione di ntJobsOS che contiene la classe unica di funzionamento acJobsOS/ncJobsOS, le funzioni di istanza di jData e Start(), la Run()di esecuzione del Loop di attesa file di comandi da remoto (jobs.ini)
    - ntJobsOS.py: Versione ancora da completare con molte classi scritta in python non da AI
        - Utilizza solo la libreria nlSys.py (condivisa con le ntJobsApp) come libreria di supporto e tutte le librerie Python standard richieste dal sistema di generazione dell’applicazione.
    - aiJobsOS.py: Versione generata totalmente tramite AI .
        - Utilizza solo la libreria aiSys.py (condivisa con le ntJobsApp) come libreria di supporto e tutte le librerie Python standard richieste dal sistema di generazione dell’applicazione.
- ntJobsOS/aiJobsOS sono una applicazione che non viene modificata, solo aggiornata integralmente, mentre la configurazione ed aggiornamento delle applicazioni e utenti riconosciute avviene cambiando il file ntjobs_config.ini e i file ntjobs*.csv degli utenti/gruppi/applicazioni, con riavvio di ntJobsOS quando vengono cambiati questi dati all’interno del normale “ciclo di lavoro” dell’orchestratore.

- L’orchestratore funziona con un ciclo elaborativo con cui cerca i file di innesco ntjobs.ini o arrivano via API, verifica l’utente che li manda, li esegue e ritorna i risultati all’utente tramite il canale dedicato con lui (share di rete).

5.6. Similitudini con IBM MQ / ApacheNifi – Polling Events

In questa modalità:

- Utente/app deposita file nella **cartella di input** su share di rete (use pattern *tmp->rename* per atomicità).
- Un **demone/servizio** (Linux systemd / Windows Service / Container) controlla la cartella ogni 15 minuti.
- Quando trova file nuovi, valida nome/formato, sposta il file in una **cartella di lavoro** e **enqueue** un job (o lo esegue subito).
- Un **worker** esegue il *guided flow* verso l’applicazione host (es.: chiamate a programmi host, script tn3270, HLLAPI, o via middleware IBM).
- Risultato → file di output in cartella di output / ingresso in MQ / notifica.
- Monitoring / logging / alerting / retry.
- Questo pattern è comune e supportato da strumenti di integrazione (NiFi, IBM MQ MFT, RPA come UiPath,

5.7. Ed in particolare similitudine con Apache NiFi

https://nifi.apache.org/nifi-docs/administration-guide.html#how-to-install-and-start-nifi

Dove è in corso la prova per far funziona/integrare **una ntjobsapp e i dati di ntJobsOs** con Apache Nifi

6. Installazione, Utilizzo e Caratteristiche di ntJobsOs

Questo capitolo interessa solo a chi vuole affiancare o controllare remotamente una ntjobsapp. ntJobsOS è lo strumento per questo. Ma se non c’è questo scopo, ma solo eseguire le ntjobsapp localmente, si può saltare questo capitolo.

6.1. Introduzione a ntJobsOS

Riepilogo: ntJobsOS trasforma le singole ntJobApp in una piattaforma completa di automazione enterprise, aggiungendo gestione utenti, esecuzione asincrona, accesso remoto via cartelle cloud e tracce di audit complete, mantenendo al contempo la semplicità e portabilità dell'architettura batch sottostante.

ntJobsOS è una piattaforma di orchestrazione specificamente progettata per gestire ed eseguire ntJobApp in modalità automatizzata e asincrona. Sebbene le ntJobApp possano funzionare autonomamente, ntJobsOS fornisce un'infrastruttura completa per:

- **Autenticazione e autorizzazione utenti** attraverso utenti e gruppi
- **Monitoraggio di cartelle remote** (share cloud) per la sottomissione di job

- **Esecuzione sequenziale o parallela** di job batch
- **Notifica automatica** dei risultati di esecuzione via email
- **Archiviazione job** per tracciabilità e auditing

Differenze Architetturali Chiave

ntJobApp	ntJobsOS
Singola micro-applicazione che esegue comandi specifici	Piattaforma che orchestra multiple ntJobApp
Può funzionare autonomamente	Funziona come demone/servizio che monitora cartelle
Accetta file .ini come parametro	Rileva automaticamente file jobs.ini in cartelle monitorate
Nessuna autenticazione utente	Sistema completo di autenticazione utenti/gruppi
Restituisce file .end	Gestisce ciclo di vita completo del job (ingestion → esecuzione → notifica → archiviazione)

6.2. Come ntJobsOS Controlla le ntJobApp

6.2.1. Flusso di Esecuzione

L'interazione tra ntJobsOS e le ntJobApp segue questo workflow:

1. **Sottomissione Job**: L'utente deposita un file `jobs.ini` nella propria cartella cloud monitorata
2. **Rilevamento**: Il demone ntJobsOS (tramite metodo `Search()`) scansiona le cartelle configurate ogni ciclo
3. **Ingestion**: ntJobsOS sposta il job in una cartella di lavoro (`Inbox`) con ID univoco basato su timestamp
4. **Autenticazione**: Valida USER e PASSWORD dalla sezione `[CONFIG]` contro `ntjobs_users.csv`
5. **Autorizzazione**: Verifica che l'utente abbia permesso di eseguire le ACTION richieste
6. **Esecuzione**: Per ogni sezione job in `jobs.ini`:
    - Crea `ntjobsapp.ini` con i parametri del job
    - Esegue la ntJobApp corrispondente tramite comando `ACT_SCRIPT`
    - Attende il file `ntjobsapp.end` (con timeout)
    - Integra i risultati nel tracking principale del job
7. **Completamento**: Crea file `jobs.end` con tutti i risultati di esecuzione
8. **Notifica**: Invia email all'utente con il riepilogo dell'esecuzione
9. **Archiviazione**: Sposta la cartella del job completato nella directory `Archive`

6.2.2. Protocollo di Comunicazione

ntJobsOS comunica con le ntJobApp attraverso un protocollo standardizzato basato su file:

Input alla ntJobApp (`ntjobs.ini`):

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
EXIT=TRUE
EXPAND=TRUE
; ... configurazione unita da ntJobsOS ...
```

```
[JOB_ID]
ACTION=nome_azione
FILE.ID1=datafile.csv
PARAM.ID1=valore1
PARAM.ID2=valore2
```

Output dalla ntJobApp (`ntjobs.end`):

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
EXIT=TRUE
EXPAND=TRUE

[JOB_ID]
ACTION=nome_azione
RETURN.TYPE=
RETURN.VALUE=Esecuzione completata con successo
TS.START=20251226.143022
TS.END=20251226.143045
FILE.RETURN.01=report_output.pdf
FILE.RETURN.02=dati_elaborati.csv
```

6.2.3. Registrazione delle Action

Affinché una ntJobApp sia richiamabile da ntJobsOS, deve essere registrata nel file in `ntjobs_actions.csv`:

```
ACT_ID;ACT_NAME;ACT_GROUPS;ACT_SCRIPT;ACT_ENABLED;ACT_PATH;ACT_TIPS;ACT_HELP;ACT
_LIVE
pdf.convert;Convertitore PDF;users;$PYTHON pdfconvert.py
ntjobsapp.ini;1;$SYSROOT/Apps;Converte documenti in PDF;FILE.INPUT=documento
sorgente;1
excel.report;Generatore Report Excel;admin,users;$PYTHON excelgen.py
ntjobsapp.ini;1;$SYSROOT/Apps;Genera report Excel;TEMPLATE=template report;1
```

Descrizione dei Campi:

- `ACT_ID`: Identificatore univoco dell'azione (usato come valore ACTION in jobs.ini)
- `ACT_NAME`: Nome leggibile dell'azione
- `ACT_GROUPS`: Lista separata da virgole dei gruppi autorizzati ad usare questa azione
- `ACT_SCRIPT`: Riga di comando da eseguire (supporta espansione $variabili)
- `ACT_ENABLED`: 1=abilitata, 0=disabilitata
- `ACT_PATH`: Directory di lavoro per l'esecuzione
- `ACT_TIPS`: Breve descrizione per gli utenti
- `ACT_HELP`: Documentazione dettagliata dei parametri
- `ACT_LIVE`: 1=bloccarla dopo un periodo di timeout, 0=non monitorare esecuzione applicazione

6.3. Configurazione di un'Installazione ntJobsOS

6.3.1. Struttura delle Directory

Un'installazione tipica di ntJobsOS richiede questa struttura di cartelle:

```
/ntJobsOS/
├── aiJobsOS.py              # Applicazione orchestratore principale
├── acJobsOS.py              # Implementazione classe core
├── aiSys.py                 # Libreria funzioni di supporto
├── nlSys.py                 # Layer di astrazione sistema (opzionale)
├── ntjobs_config.ini        # Configurazione globale
├── ntjobs_users.csv         # Database utenti
├── ntjobs_groups.csv        # Definizioni gruppi
├── ntjobs_actions.csv       # Registro azioni disponibili
├── Inbox/                   # Directory di lavoro per job attivi
│   └── job_20251226.143022/ # Cartelle job basate su timestamp
├── Archive/                 # Archivio job completati
│   └── job_20251226.120530/
└── Users/                   # Cartelle cloud utenti (monitorate)
    ├── admin/
    ├── utente1/
    └── utente2/
```

6.3.2. File di Configurazione Globale (`ntjobs_config.ini`)

```
[CONFIG]
; Versione e identificazione
TYPE=NTJOBS.CONFIG.1
LOG=C:\\Logs\\ntjobsos.log

; Email amministratore
ADMIN.EMAIL=admin@azienda.com

; Configurazione SMTP per notifiche email
SMTP.SERVER=smtp.gmail.com
SMTP.USER=notifiche@azienda.com
SMTP.PASSWORD=password_app_qui
SMTP.PORT=587
SMTP.TLS=True
SMTP.SSL=False
SMTP.FROM=noreply@azienda.com

; Percorsi di sistema (supporta espansione $variabili)
SYSROOT=C:\\ntJobsOS
ARCHIVE=$SYSROOT\\Archive
INBOX=$SYSROOT\\Inbox
GDRIVE=$SYSROOT\\Users

; Percorso interprete Python
PYTHON=C:\\Python310\\python.exe

; Impostazioni di esecuzione
EXEC=SINGLE          ; SINGLE=sequenziale, MULTI=parallelo
CYCLE.WAIT=15        ; Secondi tra scansioni cartelle
TIMEOUT=300          ; Timeout job predefinito in secondi
EXPAND=TRUE          ; Abilita espansione $variabili
```

6.3.3. Avvio di ntJobsOS

Windows (Prompt dei Comandi):

```
cd C:\ntJobsOS
python aiJobsOS.py
```

Windows (Come Servizio):

```
sc create ntJobsOS binPath="C:\Python310\python.exe C:\ntJobsOS\aiJobsOS.py"
start=auto
sc start ntJobsOS
```

Linux (systemd):

```
# Crea /etc/systemd/system/ntjobsos.service
[Unit]
Description=ntJobsOS Batch Orchestrator
After=network.target

[Service]
Type=simple
User=ntjobs
WorkingDirectory=/opt/ntJobsOS
ExecStart=/usr/bin/python3 /opt/ntJobsOS/aiJobsOS.py
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target

# Abilita e avvia
sudo systemctl enable ntjobsos
sudo systemctl start ntjobsos
```

6.4. Operazioni di Manutenzione del Sistema

6.4.1. Aggiunta di Nuovi Utenti

Modifica `ntjobs_users.csv` (delimitato da punto e virgola):

```
USER_ID;USER_PASSWORD;USER_NAME;USER_NOTES;USER_GROUPS;USER_PATHS;USER_MAIL
admin;pwd_sicura;Amministratore;Admin di
sistema;admin;$GDRIVE/admin;admin@azienda.com
mario.rossi;password123;Mario Rossi;Reparto
marketing;users;$GDRIVE/mario.rossi;mario@azienda.com
laura.bianchi;pwd456;Laura Bianchi;Reparto
finanza;users,reporting;$GDRIVE/laura.bianchi;laura@azienda.com
```

Descrizione dei Campi:

- `USER_ID`: Username univoco (usato nel campo USER di jobs.ini)
- `USER_PASSWORD`: Password in chiaro (considera crittografia in produzione)
- `USER_NAME`: Nome completo visualizzato
- `USER_NOTES`: Note amministrative
- `USER_GROUPS`: Appartenenze a gruppi separate da virgola
- `USER_PATHS`: Percorsi cartelle monitorate separati da virgola (supporta $variabili)
- `USER_MAIL`: Indirizzo email per le notifiche

Dopo la modifica: Inviare un comando di reload a ntJobsOS (vedi sezione 5.6.2)

6.4.2. Gestione dei Gruppi Utenti

Modifica `ntjobs_groups.csv`:

```
GROUP_ID;GROUP_NAME;GROUP_NOTES
admin;Amministratori;Accesso completo al sistema
users;Utenti Standard;Sottomissione job base
reporting;Generatori Report;Accesso a strumenti di reportistica
finance;Team Finanza;Elaborazione dati finanziari
developers;Sviluppatori;Accesso sviluppo e testing
```

I gruppi controllano quali ACTION gli utenti possono eseguire. Il campo `ACT_GROUPS` di un'azione deve includere almeno uno dei gruppi dell'utente.

6.4.3. Aggiunta di Nuove Action

Per registrare una nuova ntJobApp con ntJobsOS, aggiungi una voce a `ntjobs_actions.csv`:

```
ACT_ID;ACT_NAME;ACT_GROUPS;ACT_SCRIPT;ACT_ENABLED;ACT_PATH;ACT_TIPS;ACT_HELP;ACT
_LIVE
data.merge;Fusione File Dati;users;$PYTHON $SYSROOT/Apps/datamerge.py
ntjobsapp.ini;1;$SYSROOT/Apps;Unisce più file CSV/Excel;FILE.INPUT1=primo file,
FILE.INPUT2=secondo file, FORMAT=formato output (CSV/XLSX);1
```

Best Practices:

1. Usa `ACT_ID` descrittivi con prefisso di categoria (es. `pdf.`, `excel.`, `data.`)
2. Imposta `ACT_ENABLED=0` durante i test, `1` per produzione
3. Usa `ACT_LIVE=0` per funzionalità beta visibili solo agli admin
4. Fornisci `ACT_HELP` chiaro documentando i parametri richiesti
5. Usa sempre percorsi assoluti o espansione $variabili in `ACT_SCRIPT`

6.4.4. Ricaricamento della Configurazione

Dopo aver modificato qualsiasi file di configurazione (`.csv` o `.ini`), ntJobsOS deve ricaricare la sua configurazione. Questo viene fatto sottomettendo un'azione di sistema speciale (vedi sezione 5.6.2).

6.5. Sottomissione Job Remoti via Cartelle Cloud

6.5.1. Workflow Utente

Gli utenti interagiscono con ntJobsOS senza accesso diretto al server:

1. **Prepara file job**: Crea `jobs.ini` con le azioni e parametri desiderati
2. **Prepara file dati**: Posiziona tutti i file di input necessari nella stessa cartella
3. **Sottometti job**: Copia `jobs.ini` e file dati nella tua cartella cloud monitorata
4. **Attendi il completamento**: ntJobsOS elabora il job in modo asincrono
5. **Ricevi notifica**: Notifica email con i risultati dell'esecuzione
6. **Recupera output**: Tutti i file generati appaiono nella tua cartella cloud
7. **Rivedi risultati**: Controlla il file `jobs.end` per il log dettagliato dell'esecuzione

6.5.2. Esempio di Sottomissione Utente

Cartella: **`Users/mario.rossi/`** (monitorata da ntJobsOS)

Posiziona questi file:

- `jobs.ini` (specifiche del job)
- `dati_vendite.xlsx` (dati di input)

Contenuto jobs.ini:

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
USER=mario.rossi
PASSWORD=password123
EXIT=TRUE
EXPAND=TRUE

[GENERA_REPORT]
ACTION=excel.report
FILE.INPUT=dati_vendite.xlsx
TEMPLATE=vendite_mensili
OUTPUT.FORMAT=PDF
PARAM.MESE=Dicembre
PARAM.ANNO=2025
```

Risultato: Dopo l'elaborazione, l'utente trova nella sua cartella:

- `jobs.end` (log di esecuzione con timestamp)
- `report_vendite_mensili_202512.pdf` (output generato)
- Notifica email con riepilogo

6.5.3. Operazioni File Atomiche

Per prevenire che ntJobsOS rilevi file incompleti durante l'upload, usa il **pattern tmp-rename**:

```
# Esempio Python
import shutil
import os

# Scrivi su file temporaneo
shutil.copy('jobs.ini', 'Users/mario.rossi/jobs.ini.tmp')
shutil.copy('dati.xlsx', 'Users/mario.rossi/dati.xlsx.tmp')

# Rinomina atomica - solo ora visibile a ntJobsOS
os.rename('Users/mario.rossi/jobs.ini.tmp', 'Users/mario.rossi/jobs.ini')
os.rename('Users/mario.rossi/dati.xlsx.tmp', 'Users/mario.rossi/dati.xlsx')
```

Questo garantisce che ntJobsOS elabori solo file completi e completamente scritti.

6.6. Azioni di Controllo del Sistema

6.6.1. Action Interne

ntJobsOS include azioni interne speciali che non eseguono script esterni ma controllano la piattaforma stessa. Queste sono inizializzate in `asActionsInternal` e gestite dal metodo `JobInternal()`.

6.6.2. Ricaricamento Configurazione (sys.reload)

Per ricaricare tutti i file di configurazione senza fermare ntJobsOS:

jobs.ini:

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
USER=admin
PASSWORD=password_admin
EXPAND=TRUE

[RICARICA_CONFIG]
ACTION=sys.reload
```

Effetto:

- ntJobsOS crea il file marker `ntJobsOS.reload`
- La piattaforma ricarica `ntjobs_config.ini`, `ntjobs_users.csv`, `ntjobs_groups.csv`, `ntjobs_actions.csv`
- I job attivi continuano senza interruzione
- I nuovi job usano la configurazione aggiornata

Casi d'uso:

- Dopo aver aggiunto nuovi utenti o gruppi
- Dopo aver modificato definizioni di azioni
- Dopo aver aggiornato le impostazioni SMTP
- Dopo aver cambiato i percorsi delle cartelle monitorate

6.6.3. Arresto Controllato (sys.quit)

Per fermare ntJobsOS in modo controllato:

jobs.ini:

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
USER=admin
PASSWORD=password_admin
EXPAND=TRUE

[FERMA_PIATTAFORMA]
ACTION=sys.quit
```

Effetto:

- ntJobsOS crea il file marker `ntJobsOS.quit`
- Il job corrente viene completato prima dell'arresto
- La piattaforma esce in modo pulito
- Lo script launcher esterno può rilevare questa condizione

Casi d'uso:

- Prima di manutenzione del sistema
- Prima di distribuire aggiornamenti di ntJobsOS
- Arresto di emergenza quando vengono rilevati problemi

6.6.4. Riavvio Sistema (sys.shutdown)

Per innescare un riavvio completo del sistema:

jobs.ini:

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
USER=admin
PASSWORD=password_admin
EXPAND=TRUE

[RIAVVIA_SISTEMA]
ACTION=sys.shutdown
```

Effetto:

- ntJobsOS crea il file marker `ntJobsOS.shutdown`
- Il launcher esterno deve gestire il comando effettivo di riavvio del SO
- Tipicamente usato con finestre di manutenzione programmate

Attenzione: Assicurarsi che tutti i job critici siano completati prima di emettere questo comando.

6.6.5. Esempio Completo: Workflow di Manutenzione

Scenario: L'amministratore deve aggiungere nuovi utenti e azioni, poi ricaricare la configurazione

Passo 1: Modifica file di configurazione

```
# Aggiungi a ntjobs_users.csv
nuovoutente;temppass;Nuovo Utente;Assunzione Q4
2025;users;$GDRIVE/nuovoutente;nuovoutente@azienda.com

# Aggiungi a ntjobs_actions.csv
custom.import;Import Dati Personalizzato;users,admin;$PYTHON
$SYSROOT/Apps/customimport.py ntjobsapp.ini;1;$SYSROOT/Apps;Importa formato dati
personalizzato;FILE.INPUT=file dati;1
```

Passo 2: Crea job di reload

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
USER=admin
PASSWORD=pwd_admin_sicura
EXPAND=TRUE

[RICARICA_SISTEMA]
ACTION=sys.reload
```

Passo 3: Sottometti job di reload

- Posiziona `jobs.ini` nella cartella `Users/admin/`
- Attendi la conferma `jobs.end`
- La nuova configurazione è ora attiva

Passo 4: Testa la nuova configurazione

```
[CONFIG]
TYPE=NTJOBS.APP.1.0
USER=nuovoutente
PASSWORD=temppass
EXPAND=TRUE

[TEST_NUOVA_AZIONE]
ACTION=custom.import
FILE.INPUT=test_dati.csv
```

6.7. Argomenti Avanzati

6.7.1. Coordinamento Job Multi-Utente

Funzionalità futura. Esecuzione dei comandi delle ACTION in modalità parallela e non sequenziale.

EXEC=SINGLE (Sequenziale):

- Job elaborati uno alla volta, in ordine di rilevamento
- Nessun conflitto di risorse garantito
- Tempo totale di esecuzione più lungo

EXEC=MULTI (Parallelo):

- Più job possono eseguire concorrentemente
- Throughput più veloce
- Richiede ntJobApp thread-safe

6.7.2. Gestione Timeout Job

Ogni job ha un timeout configurabile:

```
[CONFIG]
TIMEOUT=600  ; 10 minuti
EXPAND=TRUE

[JOB_LUNGO]
ACTION=data.process
TIMEOUT=1800  ; Override: 30 minuti per questo job
```

Se il timeout scade:

- Il processo ntJobApp viene terminato (se possibile)
- Viene impostato `RETURN.TYPE=E` con errore di timeout
- Possono essere restituiti risultati parziali
- L'utente riceve notifica email di timeout

6.7.3. Recupero da Errori

Quando un job fallisce:

1. I dettagli dell'errore vengono catturati in `RETURN.VALUE`
2. `RETURN.TYPE=E` marca la condizione di errore
3. L'utente viene notificato via email con la descrizione dell'errore
4. I file del job vengono spostati in Archive con marker `.error`
5. L'amministratore riceve copia della notifica di errore (se configurato)

6.7.4. Traccia di Audit

Tutta l'esecuzione dei job viene registrata automaticamente:

- Il `jobs.ini` originale non viene salvato in Archive per non salvare anche le credenziali utente.
- `jobs.end` con timeline completa dell'esecuzione viene salvato in Archive
- Le notifiche email creano una traccia documentale
- I nomi delle cartelle basati su timestamp abilitano il tracking cronologico

6.7.5. Integrazione con Sistemi Esterni

ntJobsOS può innescare workflow esterni:

```
# ntjobs_actions.csv
api.webhook;Innesca Webhook;users;$PYTHON $SYSROOT/Apps/webhook.py
ntjobsapp.ini;1;$SYSROOT/Apps;Chiama API esterna;URL=endpoint, METHOD=POST/GET;1
mq.publish;Pubblica su Message Queue;admin;$PYTHON $SYSROOT/Apps/mqpublish.py
ntjobsapp.ini;1;$SYSROOT/Apps;Invia a IBM MQ/RabbitMQ;QUEUE=nome coda,
MESSAGE=payload;1
```

Questo permette a ntJobsOS di servire come ponte tra sottomissioni utente basate su file e sistemi moderni basati su API.

6.8. Confronto: Autonoma vs. Orchestrata

Aspetto	ntJobApp Autonoma	Orchestrata con ntJobsOS
Esecuzione	Linea di comando manuale	Rilevamento ed esecuzione automatici
Autenticazione	Nessuna	Validazione user/password
Autorizzazione	Implicita (chi la esegue)	Permessi azioni basati su gruppi
Input	Percorso diretto file .ini	Deposita file in cartella monitorata
Notifica	Controllo manuale file .end	Notifica email automatica
Schedulazione	Scheduler esterno (cron/Task Scheduler)	Monitoraggio cartelle integrato
Multi-utente	Non supportato	Multi-utente completo con cartelle isolate
Audit	Logging manuale	Archiviazione automatica con timestamp
Accesso Remoto	Richiede accesso server	Cartella cloud (nessun accesso server necessario)

Aspetto	ntJobApp Autonoma	Orchestrata con ntJobsOS
Caso d'Uso	Sviluppo, testing, automazione personale	Produzione, multi-utente, enterprise

6.9. Best Practices

1. **Sicurezza**
   - Usa password forti in `ntjobs_users.csv`
   - Considera la crittografia dello storage password
   - Limita l'appartenenza al gruppo `admin`
   - Rivedi regolarmente i permessi delle azioni
   - Monitora la cartella Archive per attività sospette
2. **Performance**:
   - Imposta valori `TIMEOUT` appropriati per tipo di azione
   - Usa `EXEC=MULTI` solo con azioni thread-safe
   - Pulisci periodicamente la cartella Archive
   - Monitora la dimensione della cartella Inbox
3. **Affidabilità**:
   - Esegui ntJobsOS come servizio di sistema con auto-restart
   - Configura notifiche email per avvisi admin
   - Testa nuove azioni in modalità `ACT_LIVE=0` prima
   - Mantieni i file di configurazione sotto controllo versione
   - Mantieni backup della cartella Archive
4. **Usabilità**:
   - Fornisci documentazione `ACT_HELP` chiara per tutte le azioni
   - Usa valori `ACT_NAME` descrittivi
   - Crea file `jobs.ini` template per task comuni
   - Documenta i formati file richiesti nell'help delle azioni
   - Invia job di test prima del rollout in produzione
5. **Manutenzione**:
   - Usa `sys.reload` invece di riavvii completi quando possibile
   - Schedula pulizia regolare di Archive
   - Rivedi i log per colli di bottiglia di performance
   - Aggiorna regolarmente Python e le librerie
   - Testa i cambiamenti di configurazione in ambiente isolato prima