Progetto di micro–datacenter domestico con Raspberry Pi

NAS ridondante + Web server + cluster di trading bot

Versione: 1.0 – Appunti di progetto

---

0. Cosa ripassare prima di mettere le mani sull'hardware

L'idea è costruire qualcosa che, concettualmente, è un piccolo data center: storage condiviso, servizi esposti su Internet e nodi di calcolo specializzati. Per farlo in modo solido è utile avere ben in testa alcuni concetti (qui li elenco, non li spiego in dettaglio: l'obiettivo è ricordare cosa andare a riprendere).

0.1 Networking

- Modello TCP/IP (livelli: link, IP, trasporto, applicazione).
- Indirizzamento IPv4, subnet /24, differenza tra IP privati e pubblici.
- DHCP vs IP statici; prenotazioni DHCP lato router.
- NAT casalingo e port forwarding (inbound verso un host interno).
- DNS "normale", mDNS (`hostname.local`), file `/etc/hosts`.
- Differenza tra traffico **in uscita verso Internet** (outbound) e **in ingresso da Internet** (inbound).

0.2 Sistemi operativi e sicurezza

- Gestione utenti e permessi in Linux (in particolare file con dati sensibili).
- SSH con chiave pubblica e disabilitazione del login via password.
- Firewall host-based (`ufw` o equivalente) e regole per rete locale.
- Concetto di "servizi esposti" vs "servizi solo interni alla LAN".

0.3 Storage e condivisione

- Filesystem Linux (EXT4) e concetto di mount.
- Differenza tra device (`/dev/sda`) e UUID; `/etc/fstab`.
- Concetti base di RAID, in particolare RAID1 (mirror).
- Protocolli di condivisione file:
  - SMB/Samba per PC (Windows/macOS/Linux),
  - NFS per host Linux (gli altri Raspberry).

0.4 Servizi e containerizzazione

- Concetti di base di Docker:
  - immagine, container, volume, rete.
- Principio di separazione dei servizi:
  - reverse proxy,

  - webapp,
  - bot Telegram,
  - trading bot.
- HTTPS/TLS e Let's Encrypt (o equivalenti).

0.5 Architettura e scalabilità

- Separazione dei ruoli tra nodi (storage, front-end, back-end).
- Scalabilità orizzontale: aggiungere nodi worker identici.
- Nozioni di base di sistemi distribuiti:
  - coordinamento tra più bot che toccano lo stesso exchange,
  - limiti di rate delle API.

---

1. Obiettivi del sistema

1.1 Funzionali

1. **NAS domestico** con ridondanza minima (RAID1) per dati personali e dati del cluster.
2. **Web server** con:
   - reverse proxy,
   - webapp,
   - bot Telegram (preferibilmente via container),
   - accessibile dall'esterno via un nome di dominio e HTTPS.
3. **Cluster di nodi di trading**:
   - uno o più Raspberry dedicati a trading bot,
   - solo traffico in uscita verso gli exchange,
   - nessun servizio esposto pubblicamente.
4. **Scalabilità**:
   - possibilità di aggiungere facilmente nuovi nodi worker per il trading.
5. **Portabilità fisica**:
   - la "cassetta" con switch + Raspberry può essere spostata da una casa all'altra,
   - una volta collegata al nuovo router, la struttura interna continua a funzionare con aggiustamenti minimi lato router.

1.2 Non funzionali

- **Sicurezza**: i nodi che manipolano chiavi/API del trading non sono esposti verso Internet.
- **Affidabilità**:
  - ridondanza del disco sul NAS (RAID1),
  - separazione dei ruoli (un problema sul web server non deve toccare i bot di trading).

- **Semplicità**: niente Kubernetes o orchestratori complessi; ogni Pi ha un ruolo chiaro.
- **Costo ragionevole**: usare hardware consumer, ma con una logica pulita.
- **Apprendibilità**: usare componenti e pattern che si possono riutilizzare altrove (non soluzioni troppo "magiche").

---

2. Architettura logica

2.1 Ruoli dei nodi

- `rpi-nas` – **NAS**
  - Espone lo storage ridondato (RAID1) alla LAN.
  - Fornisce:
    - condivisioni per PC (Samba),
    - condivisioni per gli altri Raspberry (NFS).
  - È il "punto di verità" per log, backup, file multimediali, ecc.
- `rpi-web` – **Web / ingress**
  - Unico nodo che riceve traffico HTTP/HTTPS dall'esterno.
  - Esegue:
    - un reverse proxy (Caddy / Nginx / Traefik),
    - una o più webapp,
    - bot Telegram.
  - È protetto da firewall, ma è l'unico nodo con port forwarding dal router.
- `rpi-trade-1, rpi-trade-2, ...` – **worker di trading**
  - Eseguono uno o più trading bot ciascuno.
  - Devono poter **uscire** verso Internet (API exchange), ma non devono avere porte aperte dall'esterno.
  - Possono scrivere log sul NAS e leggere eventuali dati comuni.

2.2 Topologia di rete (singola LAN)

Tutto è dietro un unico router casalingo e un unico switch gigabit:

- Router del provider (es. 192.168.1.1) con NAT e DHCP.
- Switch gigabit non gestito collegato alla porta LAN del router.
- Sullo switch:
  - `rpi-nas`,
  - `rpi-web`,
  - `rpi-trade-*`,
  - eventuali PC/altro.

La separazione non è tramite reti diverse, ma tramite: - **scelta di quali porte il router espone verso Internet** (solo 80/443 verso `rpi-web`), - **firewall host-based** su ogni Raspberry.

2.3 Indirizzamento e nomi

- Ogni Raspberry ha un **hostname significativo** (`rpi-nas`, `rpi-web`, `rpi-trade-1`, …).
- Il router assegna gli IP via **DHCP**; idealmente si impostano **reservation** per avere IP stabili (es. 192.168.1.10 per NAS, .11 per web, .12 per trade-1, ecc.).
- Nei servizi interni si usano **hostname** (o `hostname.local` via mDNS) e non IP hardcodati, per facilitare lo spostamento del cluster su router diversi.

2.4 Spostabilità tra case diverse

Quando il cluster viene portato in un'altra casa:

- dal punto di vista interno, la logica non cambia: i Raspberry continuano a parlarsi per hostname;
- il nuovo router assegna nuovi IP (o si riconfigurano le reservation);
- occorre ripristinare sul nuovo router:
  - il port forwarding verso `rpi-web`,
  - eventuale configurazione di Dynamic DNS / record DNS.

Se si vuole ridurre al minimo i cambi di configurazione ogni volta, conviene: - usare sempre lo **stesso range di LAN** (es. forzare il router a 192.168.1.0/24 in tutte le case), - evitare IP statici configurati direttamente nei Pi (lasciare tutto al DHCP/router).

---

3. Componenti da acquistare e requisiti hardware

Le cifre sono indicative, basate su prezzi tipici in Italia a fine 2025 (range, non valori puntuali).

3.1 Nodo NAS (`rpi-nas`)

Obiettivo: storage centrale con ridondanza minima RAID1.

Componenti consigliati:

- Raspberry Pi 4 (4 GB RAM):
  - fascia indicativa: 70–90 € per la sola scheda, a seconda del rivenditore e della disponibilità.
- Due dischi per il RAID1:
  - opzione HDD 3,5" da 4 TB di classe desktop/NAS:
    - ordine di grandezza: 65–110 € ciascuno per modelli 4 TB di marche comuni (WD, Seagate).

  - alternativa SSD 2,5" da 2 TB (più silenziosi, meno capienza a parità di costo):
    * fascia tipica: 80–130 € ciascuno per SSD SATA 2 TB di marca.
- Due box USB 3.0 per dischi:
  - se si usano HDD 3,5", servono **box con alimentazione propria** (i 3,5" non possono essere alimentati solo via USB).
  - budget: circa 25–40 € per box singolo decente.
  - in alternativa un box dual-bay, ma è più costoso e meno modulare.
- Alimentatore ufficiale o equivalente per il Raspberry.
- microSD 16–32 GB per il sistema operativo (8–12 €).
- Case e, se possibile, dissipatore/ventola silenziosa: 10–20 €.

Perché due dischi identici: semplifica gestione del RAID1 e sostituzione in caso di guasto; usare dischi di taglio e prestazioni simili evita strozzature.

3.2 Nodo Web (`rpi-web`)

Obiettivo: front-end esposto a Internet per webapp e bot Telegram.

Componenti consigliati:

- Raspberry Pi 3B+ o 4 (2–4 GB RAM):
  - 3B+: ~40–60 € se reperibile,
  - 4 con 4 GB: 70–90 €.
- microSD 16–32 GB per sistema e container.
- Case con dissipazione adeguata (il traffico HTTPS tende a scaldare un po' se il carico cresce).
- Connessione Ethernet verso lo switch (evitare il Wi-Fi sul nodo esposto).

3.3 Nodi Trading (`rpi-trade-*`)

Obiettivo: eseguire trading bot isolando il più possibile i segreti dai nodi esposti.

Componenti consigliati per ogni nodo:

- Raspberry Pi 3B+ o 4 con almeno 2 GB RAM (trading bot raramente sono pesantissimi, ma dipende dall'implementazione).
  - budget: 50–80 € a nodo, a seconda del modello e del mercato del momento.
- microSD 16–32 GB.
- Case base con un minimo di ventilazione (specie se il bot lavora h24).
- Connessione Ethernet se possibile (stabilità).

Perché più nodi: - distribuire il carico su coppie/timeframe/strategie diverse, - aumentare la resilienza (se un Pi va giù, gli altri continuano).

3.4 Rete e accessori comuni

- Switch gigabit non gestito 8 porte:

  - prodotti base (es. Netgear GS308 o equivalenti) si trovano tipicamente nell'ordine dei 20–30 €.
- Cavi Ethernet Cat 5e o 6 per collegare tutti i dispositivi allo switch:
  - prevedere qualche cavo in più per flessibilità.
- Eventuali ciabatte di alimentazione di buona qualità (tutti i Raspberry + box dischi).
- Facoltativo ma consigliato: piccolo UPS per almeno NAS + switch + router, per evitare crash in caso di blackout brevi.

3.5 Stima dei costi (cluster minimo)

Configurazione minima ragionevole:

- 1 × `rpi-nas` (Pi 4 4GB) + 2 × HDD 4TB + 2 box USB + SD + case:
  - ordine di grandezza: 320–380 €.
- 1 × `rpi-web` (Pi 4 4GB o simile) con SD + case:
  - 90–110 €.
- 1 × `rpi-trade-1` con SD + case:
  - 70–90 €.
- Switch 8 porte + cavi:
  - 40–50 € considerando un po' di cablaggio extra.

Totale indicativo per: NAS + Web + 1 nodo di trading **520–630 €**.

Ogni nodo di trading aggiuntivo: ~70–90 € (solo board, SD, case, cavo).

Queste cifre sono volutamente conservative e orientate ad hardware nuovo; è possibile risparmiare acquistando usato o riutilizzando componenti già disponibili.

---

4. Storage: progettazione della NAS e RAID1

4.1 Perché un NAS separato

Separare lo storage su un nodo dedicato (`rpi-nas`) ha vari vantaggi:

- Il volume RAID1 è gestito una sola volta e condiviso con tutti gli altri nodi.
- Un problema sul nodo web o sui nodi di trading non ha effetti diretti sul filesystem di base dei dati.
- È più facile ragionare su backup e politiche di retention quando lo storage è centralizzato.

4.2 Perché RAID1 (e non altro)

- RAID1 offre la forma più semplice di ridondanza:
  - due dischi contenenti la stessa informazione,

  - se uno si guasta, il sistema continua a funzionare (in modalità degradata) finché non si sostituisce il disco.
- Altri livelli (RAID5, RAID10, ecc.) avrebbero senso con più dischi e controller più robusti; su un Raspberry con due dischi il RAID1 è il compromesso naturale.

Limiti da ricordare:

- RAID1 **non è un backup**:
  - se cancelli un file, si cancella su entrambi i dischi;
  - se scrivi dati corrotti o malware cifra i file, lo fa su entrambe le copie.
- Per vera protezione serve un backup separato (es. disco esterno che si attacca solo per backup periodici, o un cloud).

4.3 Scelta del tipo di disco

- **HDD 3,5"**:
  - pro: costo/GB inferiore, tagli più grandi (4 TB, 6 TB, …),
  - contro: servono box con alimentazione propria, sono più rumorosi e sensibili a urti.
- **SSD 2,5"**:
  - pro: silenziosi, consumi più bassi, nessuna parte meccanica,
  - contro: costo/GB superiore, tipicamente tagli più piccoli allo stesso budget.

Per un uso NAS domestico misto (media, backup, log), 2×4 TB HDD è un buon punto d'equilibrio; se il focus è su pochi dati ma importanti e letture/scritture random, 2×2 TB SSD può avere senso.

4.4 Layout logico sul NAS

Una possibile suddivisione interna dello storage RAID1 (una volta creato il volume e montato, ad esempio, sotto `/srv/nas`):

- `/srv/nas/home` – dati personali, backup da PC.
- `/srv/nas/cluster/logs` – log dai nodi del cluster (web, trading).
- `/srv/nas/cluster/data` – dati condivisi tra i nodi (dataset, configurazioni non sensibili).
- `/srv/nas/backup` – eventuali snapshot / backup di sistemi interni.

Questa separazione è utile per:

- applicare politiche di backup diverse per sezione,
- differenziare permessi di accesso (ad esempio, i bot di trading non hanno bisogno di vedere tutto).

4.5 Esportazione dei dati

- Verso PC:

  - usare Samba/SMB,
  - esporre condivisioni con utenti/password separati (es. utente `nasuser`).
- Verso altri Raspberry:
  - usare NFS per directory specifiche,
  - esportare sulla sola subnet LAN (es. 192.168.1.0/24) e proteggere con firewall.

Motivo: NFS è nativamente integrato negli strumenti Linux e, in un contesto di cluster, è più naturale da automatizzare e monitorare rispetto a Samba.

4.6 Perché software RAID e non box RAID "magico"

Una alternativa sarebbe usare un box USB che implementa RAID1 a livello hardware e presentare al Pi un singolo disco logico. È certamente più semplice da configurare, ma:

- il diagnostico (stato del mirror, rebuild, errori) dipende dal firmware del box,
- in caso di guasto dell'elettronica del box, recuperare i dischi può essere più complesso.

Con un RAID1 software su Linux:

- lo stato dell'array è visibile e scriptabile,
- la logica di rebuild e sostituzione disco è documentata,
- è più facile migrare a un'altra macchina se un giorno si abbandona il Raspberry.

---

5. Nodo Web: esposizione verso Internet in modo controllato

5.1 Ruolo del nodo Web

`rpi-web` è l'unico punto in cui il mondo esterno tocca il tuo cluster:

- riceve traffico HTTP/HTTPS sulla porta 80/443 tramite port forwarding del router,
- ospita i servizi "pubblici":
  - reverse proxy,
  - eventuale webapp,
  - bot Telegram (con webhook o polling).

Tutti gli altri nodi non hanno porte esposte su Internet; da fuori esiste, concettualmente, **un solo ingresso**.

5.2 Perché usare un reverse proxy

Il reverse proxy (Caddy, Nginx, Traefik, ecc.) si occupa di:

- terminare TLS (Let's Encrypt, gestione certificati),
- instradare le richieste in base al dominio e al path verso i diversi servizi interni (webapp, webhook dei bot),
- offrire un solo punto di hardening (limiti, headers di sicurezza, logging centralizzato del traffico HTTP).

In pratica, `rpi-web`:

- espone solo 80/443 al mondo,
- parla con servizi interni (spesso su porte non esposte) sulla stessa macchina o sulla LAN,
- mantiene i container applicativi il più isolati possibile.

5.3 Containerizzazione dei servizi Web e dei bot

Eseguire webapp e bot Telegram in container ha diversi vantaggi:

- packaging consistente tra nodi (immagine Docker uguale ovunque),
- possibilità di aggiornare/rollback rapido,
- isolamento delle dipendenze (librerie Python/Node/altro non contaminano il sistema base).

Dal punto di vista operativo, l'obiettivo non è collezionare file di configurazione, ma arrivare a:

- un file di definizione dello stack (anche solo mentalmente),
- un pattern in cui il reverse proxy parla con servizi identificati da nome (es. "webapp", "telegram-bot") senza ancorarsi a IP/porta manuali.

---

6. Nodi di trading: sicurezza e scalabilità

6.1 Perché separare i nodi trading dal nodo Web

I nodi di trading sono quelli che custodiscono le API key verso gli exchange. Tenerli separati da ciò che è direttamente esposto a Internet (nodo web) riduce:

- la probabilità che un problema lato web comprometta le chiavi di trading,
- la superficie d'attacco complessiva (ogni nodo trading non ha porte forwardate dal router).

Concettualmente:

- `rpi-web`: "facciata" esposta, ma senza accesso diretto ai segreti di trading,
- `rpi-trade-*`: "motore" di trading, vedibile solo dalla LAN, con accesso in uscita verso l'exchange.

6.2 Politica di rete per i nodi trading

- Nessun port forwarding dal router verso questi IP.
- Firewall locale parametrizzato su:
  - **incoming**: solo dalla LAN (SSH e, se serve, NFS/Samba),
  - **outgoing**: consentito per chiamare API degli exchange, Telegram (se i bot parlano direttamente), o altri endpoint necessari.

Significato del "non deve accedere a Internet" correttamente interpretato:

- il bot **deve** raggiungere Internet (outgoing) per funzionare,
- il nodo **non deve essere raggiungibile da Internet** (no ingress diretto).

6.3 Gestione dei segreti

Gli elementi sensibili (API key, secret, eventuali token) devono:

- risiedere su file dedicati sul nodo di trading,
- avere permessi stretti (lettura solo dell'utente che esegue il bot),
- non essere salvati sul NAS in chiaro (il NAS è condiviso e, per definizione, più "ampio").

Per evitare fughe accidentali:

- evitare di committare le chiavi in repository Git,
- usare file di configurazione distinti dalle immagini dei container,
- centralizzare dove possibile la logica di accesso alle chiavi (es. tutte le letture da un unico modulo del bot).

6.4 Log e osservabilità

Scrivere log significativi su NAS (es. in `/srv/nas/cluster/logs`) è utile per:

- avere una traccia centralizzata delle operazioni,
- analizzare il comportamento di più bot nel tempo,
- fare audit successivi.

Va deciso a priori cosa loggare:

- ordini inviati (senza esporre dati sensibili),
- errori delle API,
- stati interni rilevanti (apertura/chiusura posizioni, cambi di strategia, ecc.).

6.5 Aggiunta di nuovi worker

Aggiungere un nuovo nodo trading (`rpi-trade-3`, `rpi-trade-4`, …) equivale a:

- replicare lo stesso pattern:
  - stesso OS base,
  - stessa policy di firewall,

  - stesso schema di montaggio del NAS (se serve),
  - stesso modello di container per il bot,
- differenziare solo:
  - la configurazione logica (strategie, coppie, timeframe),
  - le chiavi API (se si segmentano account/permessi),
  - l'identificatore del worker (per logging e monitoraggio).

Dal punto di vista del rischio, è critico evitare che:

- due bot indipendenti agiscano sullo stesso conto e sulla stessa coppia in modo non coordinato,
- la somma del traffico generato da tutti i worker superi i limiti di rate dell'API.

Per minimizzare problemi:

- assegnare **ambiti di responsabilità ben definiti** a ciascun worker (per esempio "worker-1 → spot BTC/ETH", "worker-2 → altcoin", "worker-3 → futures"),
- eventualmente usare account separati sugli exchange per gruppi di bot diversi.

---

7. Networking: dettagli operativi e motivazioni

7.1 Assegnazione degli IP

Obiettivo: avere indirizzi stabili senza legare la configurazione dei Pi a un singolo router.

Scelta consigliata:

- lasciare i Raspberry in modalità "cliente DHCP",
- configurare sul router delle **prenotazioni** (DHCP reservation) sulla base dei MAC address,
- usare solo hostname nei riferimenti interni.

Motivo:

- se cambi router o sposti il cluster in un'altra casa, puoi replicare la stessa mappatura di IP/hostname lavorando solo lato router, senza dover toccare ogni singolo Pi.

7.2 Hostname e risoluzione dei nomi

Per la comunicazione interna tra Raspberry:

- usare hostname "parlanti" (`rpi-nas`, `rpi-web`, `rpi-trade-1`),
- fare affidamento su:

  - DNS interno del router, se lo gestisce (molti router associano automaticamente hostname IP),
  - oppure mDNS (`hostname.local`),
  - in ultima istanza, file `/etc/hosts` sui Pi per legare hostnames e IP.

Motivo: l'uso di hostname rende più robusto lo spostamento e semplifica anche la lettura delle configurazioni una volta che passano mesi.

7.3 Port forwarding

Solo `rpi-web` deve essere visibile dall'esterno. Pertanto sul router si configura port forwarding:

- porta esterna 80 → IP di `rpi-web`, porta 80,
- porta esterna 443 → IP di `rpi-web`, porta 443.

Motivo:

- avere un **singolo punto di ingresso**,
- semplificare la gestione di certificati TLS (il reverse proxy sul nodo web si occupa di tutto).

Nessun port forwarding verso:

- `rpi-nas`,
- `rpi-trade-*`,
- eventuali altri nodi di calcolo.

7.4 Firewall sui Raspberry

Il firewall host-based è l'ultima linea di difesa nel caso in cui qualcosa vada storto a livello di router o rete.

Principi generali:

- **NAS**:
  - accetta connessioni solo dalla LAN su:
    - SSH (amministrazione),
    - servizi di condivisione (Samba/NFS),
  - nessun servizio ascolta su Internet direttamente.
- **Web**:
  - accetta SSH (administration),
  - accetta HTTP/HTTPS (reverse proxy),
  - eventuali servizi interni sono confinati alla LAN o alla loopback.
- **Trading**:
  - accettano solo SSH dalla LAN,
  - non espongono altri servizi,
  - possono aprire connessioni verso l'esterno per chiamare API.

Motivo:

- se un giorno ti accorgi che il router ha una configurazione errata, il danno è mitigato da regole locali sui Pi.

---

8. Scalabilità e future evoluzioni

Una volta stabilito il pattern NAS + Web + N nodi trading, le evoluzioni naturali sono:

- **Più worker di trading**:
  - per separare strategie e mercati,
  - per distribuire carico CPU e memoria.
- **Più servizi sul nodo web**:
  - ospitare ulteriori webapp (monitoring, dashboard, grafici di trading),
  - esporre API interne per interrogare lo stato dei bot (rigorosamente dietro autenticazione).
- **Introduzione graduale di un orchestratore** (solo se ha senso):
  - quando il numero di servizi/worker cresce, strumenti come k3s/microk8s o Nomad possono diventare interessanti,
  - l'architettura di base (ruoli separati, NAS centralizzato, reverse proxy) resta valida.

È importante non introdurre complessità prematura: prima si consolida il flusso operativo con pochi nodi, poi – se il sistema diventa un laboratorio permanente – si valuta l'adozione di strumenti più sofisticati.

---

9. Roadmap operativa (alto livello)

Questa sezione riassume le macro–fasi operative, con l'enfasi sul *perché* di ogni gruppo di operazioni.

Fase 1 – Progettazione dettagliata

- Definire esattamente quali dati devono stare sul NAS (dimensionare dischi di conseguenza).
- Decidere quanti nodi trading si vogliono avere nel breve termine (es. 1–2).
- Scegliere i modelli di Raspberry in funzione di disponibilità e budget.

Motivo: ridurre sprechi di budget e scegliere tagli di storage coerenti con le esigenze reali.

Fase 2 – Acquisto dell'hardware

- Acquistare:
  - Raspberry Pi con relative alimentazioni e case,
  - switch gigabit,

  - dischi e box USB,
  - cavi Ethernet e eventuali adattatori.
- Verificare subito all'arrivo che tutto l'hardware sia funzionante (test minimi).

Motivo: intercettare componenti difettosi rapidamente, prima di investirci ore di configurazione.

Fase 3 – Assemblaggio fisico

- Montare i Raspberry nei case, collegando eventuali ventole/dissipatori.
- Installare i dischi nei box USB e collegarli al NAS.
- Cablaggio:
  - tutti i Raspberry allo switch,
  - switch al router,
  - alimentazioni ordinate (etichettare alimentatori e cavi).

Motivo: avere un setup fisicamente solido e manutentabile, in cui sia chiaro quale cavo alimenta cosa.

Fase 4 – Bootstrap dei sistemi operativi

- Preparare le microSD con Raspberry Pi OS Lite 64-bit, impostando hostname e SSH.
- Avviare i Pi e verificare che siano raggiungibili via SSH dalla LAN.
- Aggiornare i sistemi e creare eventuali utenti dedicati (es. un utente "trader" sui nodi di trading).

Motivo: uniformare il sistema base su tutti i nodi e garantirsi un accesso remoto stabile e sicuro.

Fase 5 – Configurazione di rete

- Sul router:
  - assegnare prenotazioni DHCP per ogni Raspberry,
  - abilitare la risoluzione dei nomi se disponibile.
- Verificare che hostname e IP siano coerenti con il disegno (NAS, Web, Trading).

Motivo: avere una base stabile per script, mount di rete e monitoraggio, pensando già alla portabilità del cluster.

Fase 6 – Allestimento del NAS

- Sul `rpi-nas`:
  - inizializzare i dischi e configurare il RAID1,
  - creare il filesystem e il layout di directory previsto,
  - configurare Samba per i PC e NFS per i Raspberry,

  - impostare il firewall in modo che accetti solo traffico dalla LAN sulle porte necessarie.

Motivo: costruire il "cuore dati" del sistema in modo robusto prima di appoggiarci sopra i servizi applicativi.

Fase 7 – Nodo Web

- Sul `rpi-web`:
  - installare il runtime per i servizi (es. Docker),
  - predisporre il reverse proxy,
  - configurare il dominio e Let's Encrypt o equivalente,
  - pubblicare una prima webapp minima e, in seguito, i bot Telegram.
- Sul router:
  - configurare port forwarding 80/443 verso `rpi-web`.

Motivo: verificare da subito che la catena Internet → router → `rpi-web` funzioni correttamente, prima di introdurre componenti più delicati (trading).

Fase 8 – Nodi di trading

- Su ogni `rpi-trade-*`:
  - installare il runtime necessario (Docker o direttamente il runtime del bot),
  - configurare firewall, accesso al NAS e utenti,
  - predisporre il bot con configurazioni e chiavi API adeguate (con permessi minimi necessari sull'exchange),
  - definire un sistema di logging verso il NAS.

Motivo: attivare i bot in un ambiente già strutturato (storage, rete, logging), riducendo il rischio di configurazioni "sporche" che saranno poi difficili da ripulire.

Fase 9 – Test integrato e scenari di guasto

- Verificare:
  - accesso al NAS da PC e dagli altri Pi,
  - comportamento del sistema se un nodo trading va giù,
  - comportamento del sistema se un disco del NAS si rompe (simulabile scollegando un disco a sistema spento),
  - capacità di spostare fisicamente il cluster su un'altra rete e ripristinare la funzionalità con soli interventi sul router.

Motivo: non scoprire i comportamenti di guasto il giorno in cui qualcosa si rompe davvero, ma avere già procedure e intuizioni su cosa osservare e dove intervenire.

Questo documento non è una guida con comandi pronti da copiare–incollare, ma uno schema di progetto: definisce ruoli, motivazioni, vincoli e macro–operazioni. La parte “hands-on” (config file, script, automazione) può essere costruita sopra questi concetti, mantenendo la struttura coerente e facilmente estendibile nel tempo.