JANZEER BLOCKCHAIN

Node Operator Guide

Install · run · validate

Document v1.0 · Janzeer Network · 2026

Janzeer Node Operator Guide

This guide is for people who run Janzeer nodes and for those who want to become validators. It covers what a node does, how to **run a validator node from a prebuilt native binary** (the end-user path), configuration (ports, secrets, NTP, database, and genesis overrides), how to register as a validator (permissionlessly, behind a non-refundable deposit) using the non-custodial wallet, how to monitor and operate a node, and — for the internal team — how to build and run from source. Janzeer is a fixed-supply, **permissionless stake-free** Layer-1 — validators register with no vote and no approval, produce blocks in a fixed rotation, and finalize with two-phase BFT; the node is **non-custodial** — it verifies signatures and never holds user keys. For the economic and protocol details referenced here, see the White Paper and the Yellow Paper.

> **Janzeer is not open source, but the node is a public download.** End users do **not** build from source and do **not** run a jar: they download the native bundle for their platform (Linux, Windows, macOS, or the Docker image), verify its checksum and start it (§2). There is no JAR and no source build to run — §6 lists what is distributed.

1. Overview: what a node does

Every Janzeer node does the same baseline work: it connects to peers over an authenticated P2P link, gossips and validates blocks and transactions, syncs the chain, persists state to a database, and serves the REST API and explorer endpoints. There are two operating roles, and the difference is entirely about whether the node holds a valid signing key.

- **Watch-only (sync-only) node** — any node whose key is **not registered** on-chain. It boots, dials peers, syncs the chain, accepts transactions into its mempool, and serves the full read API and explorer. It does **not** produce blocks and does **not** cast finality votes. This is what a fresh download is, and the default for anyone who just wants a synced node or an explorer/API backend. It needs no inbound port.
- **Active validator** — a node whose validator key (the public key of the `secret` in its `config.json`) is registered and in the **active set**. It owns block-production slots (deterministic round-robin) and casts signed `PREPARE`/`COMMIT` votes. To be active, the key must be either a genesis **anchor** or a registered validator that made the active set — and selection is purely by **registration order** (longest-registered first), not by votes or rating (see §4).

A critical reminder: **a single node cannot finalize anything.** Finality is BFT two-phase (`PREPARE` → `COMMIT`) and requires a live **quorum** of the active set — `bftQuorum(N) = floor(2N/3) + 1` over the actual active-set size `N`. With the three genesis anchors, quorum is 3, so all three must be live and voting to finalize blocks. A lone node will accept

transactions into its mempool but will never commit a block by itself. Run **at least 4 validators** so the network can tolerate one fault (`N ≥ 3f+1`).

2. Run a node (public download)

Anyone can run a Janzeer node from the **public binary download** — no Java, no database server, no build. One archive per platform (Linux x86-64, Linux arm64 for the Raspberry Pi, Windows x86-64, macOS Apple silicon / Intel) plus a Docker image, all built from the same release: a native executable, its runtime libraries, the network's public `genesis.json`, a launcher, `README.txt`, `LICENSE.txt`, `VERSION` and `SHA256SUMS`. The bundle contains **no keys and no secrets**; your node creates its own identity on first start.

There are exactly two ways to run it, with the **same binary**:

Role	What it does	Needs
Watch-only (default)	Follows the chain, serves the explorer and the REST/JSON-RPC/WebSocket API on your machine, never produces or votes.	Nothing but an outbound internet connection — the node dials out to its peers.
Validator	The same node after you register its key on-chain: it produces blocks in its slots and votes on finality.	Its P2P port reachable from the internet, a wallet holding the **2000 JNZ non-refundable deposit** + the 3 JNZ fee, and exactly one node per key.

There is no delegation and no staking: you either run a node or you don't.

Step 1 — Download and verify

Get the archive for your platform from the **Validators** page on janzeer.org (or the address in `release_node.md` for the team). The page shows the SHA-256 of every archive; check it before unpacking:

```
# Linux / macOS
sha256sum -c SHA256SUMS          # or: shasum -a 256 -c SHA256SUMS
tar -xzf janzeer-node-<version>-<network>-linux-x64.tar.gz
# Windows (PowerShell)
Get-FileHash janzeer-node-<version>-<network>-windows-x64.zip   # compare with the
published value
Expand-Archive janzeer-node-<version>-<network>-windows-x64.zip
```

`<network>` is `mainnet` (the only public network). The same kit can cut a `testnet` bundle (network id `janzeer-testnet`) for a private test network you run yourself — coins there have no value; there is no public testnet. A bundle of one network can never join another: the network id is signed into every transaction and pinned in the peer handshake.

Requirements: a 64-bit OS, 1 GB of free RAM, 20 GB of disk, and an **NTP-synchronised clock** (a validator halts production if its clock is more than 250 ms off).

Step 2 — Start it

```
./start.sh          # Linux / macOS
start.cmd           # Windows
docker run -d --name janzeer -p 127.0.0.1:7019:7019 -p 9199:9199 -v janzeer-
data:/opt/janzeer/data ghcr.io/janzeerorg/janzeer-node:latest   # Docker (x86-64 +
arm64)
```

The node creates `./data` (ledger + `data/config.json`, its identity), dials the seed peers baked into the launcher, syncs the chain and serves the explorer on `http://localhost:7019`. Check progress with:

```
curl -s http://localhost:7019/api/v1/info
# …
"networkId":"janzeer","genesisHash":"…","version":"0.1.14","syncStatus":"SYNCHRONIZED"
…
```

`syncStatus` becomes `SYNCHRONIZED` when the node has caught up. That is a complete **watch-only** node.

Everything is tuned through environment variables read by the launcher (see `README.txt` in the bundle): `JANZEER_REST_PORT` (7019), `JANZEER_P2P_PORT` (9199), `JANZEER_ROOT_NODES` (seed peers), `JANZEER_DATA_DIR` (`./data`), `JANZEER_JAVA_OPTS` (`-Xmx256m`), and `DB_URL` / `DB_USERNAME` / `DB_PASSWORD` to use PostgreSQL instead of the embedded database. The Linux bundle ships a `janzeer-node.service` unit with install instructions inside.

Step 3 — Back up your identity

`data/config.json` holds the node's signing key (the **validator key** you will register) and the seed phrase of the node's own wallet. Copy it somewhere safe and keep it secret. The rest of `data/` is the ledger and can always be resynced from the network.

Step 4 — Become a validator

1. **Open the P2P port** (default `9199/tcp`) to the internet, so other nodes can dial yours. A validator behind a home router still works without it: every block, vote and certificate travels over the outbound connections your node keeps to its peers, and each node relays what it receives. The open port adds direct links to other validators and lets peers reach you when your own connections drop, so forward it when you can.
2. **Read your node key**: `curl -s http://localhost:7019/api/v1/info` → `"nodeKey": "02…"` (also shown by the explorer your node serves).
3. **From a Janzeer wallet** (web wallet at any explorer, or the mobile app): open the **Validators** tab, paste the node key and choose **Register**. The wallet signs and submits the registration transaction (`POST /api/v1/transactions/validators`) carrying the **2000 JNZ non-refundable deposit** plus the exact fee of 3 JNZ. Keys never leave the wallet; the node is non-custodial.

4. Your node is admitted to the active set at the **next epoch boundary** and starts producing in its slots. Confirm with `/api/v1/validators/active`.

From here on, §4 applies: rewards go 100 % to the producing validator's wallet, you can exit gracefully (`POST /api/v1/transactions/exit-validators`; the deposit stays non-refundable), a node silent on its slot is ejected for downtime, and you must run **exactly one node per key** — two nodes sharing a `data/config.json` get slashed for equivocation.

Step 5 — Upgrade

You are told, you decide. The node checks once a day whether a newer release exists (`node.update.url`, the project's manifest at `downloads.janzeer.org/node/latest.json`; notification only, it never downloads or installs anything; `--node.update.enabled=false` opts out). A newer release appears as one warning line in the log, a notice at the top of the explorer and in `GET /api/v1/info/update`.

Linux and macOS bundles ship `update.sh`: `./update.sh --check` reports, `./update.sh` downloads the new bundle, verifies the project's **minisign signature** on the release's `SHA256SUMS` and the archive's checksum against it, keeps `./data`, swaps the files, and restarts `janzeer-node.service` if that is how the node runs (`./update.sh --rollback` restores the previous version from `.previous/`). It needs `minisign` installed and refuses an unsigned or tampered release. The release public key (minisign, key id `9F798583B345AE01`) is `RWQBrkWzg4V5n5wUBa9GNPpa95Z7WFT2EY5QrabMv/DP+KG+gJsPIgdr`; it is printed in `update.sh`, in the bundle's `README.txt` and on https://janzeer.org/validators — compare them. To check a release by hand: `minisign -Vm SHA256SUMS -P RWQBrkWzg4V5n5wUBa9GNPpa95Z7WFT2EY5QrabMv/DP+KG+gJsPIgdr`. Windows: stop the node, unpack the new zip over the folder (keep `data\`), verify `SHA256SUMS`, start again. Arch Linux: `pacman -U` the new package.

Since 0.1.11 every bundle also ships `jz`, the command-line client, next to the `janzeer` binary: `./jz node status` (version, sync, health, update availability), `./jz node update` (exit 10 = update available, `--run` installs), `./jz watch blocks`, `./jz info`; it needs no `--node-dir` when run from the bundle folder. The full guide is the Command-Line Client Guide (`cli.md`); `update.sh` keeps it up to date with the node.

By hand on any platform: stop the node, unpack the new bundle over the folder (keep `./data`), verify `SHA256SUMS`, start again. `VERSION` lists the wire protocol version; peers on another protocol version disconnect you with `protocol version mismatch` in the log — that line means "upgrade". Validators upgrade one at a time so the network keeps its quorum, except when a release notes say all nodes restart together (a protocol bump).

Troubleshooting

Log line / symptom	Meaning	Fix
`GENESIS MISMATCH: …` and the process exits	The `genesis.json` in the folder describes another chain than the ledger in `./data` (e.g. a testnet bundle over a mainnet data directory) or the file was edited.	Use the matching bundle, or move `./data` away and resync.
`genesis file … is for network 'janzeer-testnet' but this node runs consensus.network-id 'janzeer'`	A launcher edited by hand points a mainnet node at a testnet genesis (or vice versa).	Re-extract the bundle; do not mix files across networks.
`Closing … after a decode error: protocol version mismatch`	Your node or the peer runs an older release.	Upgrade.
`Rejecting … chain-spec digest mismatch` in a peer's log	Different consensus rules or a different genesis — another chain.	Same as above: matching bundle.
`syncStatus` stays `PROCESSING`	Still catching up, or no reachable seed peer.	Wait; check `JANZEER_ROOT_NODES` and outbound connectivity.
Registered but not producing	Clock off by > 250 ms, P2P port closed, or the epoch boundary has not passed yet.	Fix NTP / firewall; wait for the next epoch.

3. Configuration

Configuration comes from built-in defaults, command-line `--key=value` overrides (the launcher passes these), environment variables, and the per-node `config.json` identity file. On a downloaded bundle you normally touch nothing: the launcher's environment variables (§2, Step 2) cover ports, seed peers, data directory and database.

Ports

Property	Default	Purpose
`server.port`	`7019`	REST API / UI
`node.port`	`9199`	P2P (Netty TCP)

`node.*` — networking and clock

Property	Default	Purpose
`node.root-nodes`	the network's seed peers (baked into the bundle launcher; `JANZEER_ROOT_NODES` overrides)	Seed/root peers dialed on startup for discovery.
`node.peers-number`	`3`	Target number of inbound peer connections.
`node.protocol-version`	`3.4.0`	P2P wire protocol version; a peer on another version is disconnected (`protocol version mismatch` in the log).
`node.ntp-servers`	(configured list)	NTP servers used for clock sync.

Clock sync is required to produce. The node halts block production if its clock is not synchronized (NTP offset must be within `ntp-offset-threshold = 250 ms` — tightened for the 15-second wall-clock slots) or if it is not fully synced. A node with a drifting clock will still serve the API and sync, but it will silently miss its production slots — keep NTP working.

`consensus.*` — economic & consensus parameters (highlights)

These parameters are the **genesis-pinned rule-set**: everything under `consensus.*` is the canonical consensus rule-set fixed at genesis. Each node computes a **chain-spec digest** (double-SHA256 over these params **and the epoch-1 genesis hash**) and exchanges it in the P2P handshake; a peer whose digest differs — different rules *or* a different genesis — is rejected at greeting. **Do not change any `consensus.*` value on a live network** — a node that edits one forks itself off the network instead of joining it. Only operational settings (ports, DB, peers, NTP) are per-node. You should still know these to operate correctly; see the Yellow Paper for the full set.

Property	Default	Meaning
`consensus.network-id`	`janzeer`	The network's identity, signed into every transaction and the handshake. A private test network uses `janzeer-testnet`.
`consensus.promoters-count`	`21`	Max size of the active set.
`consensus.rotating-producer-seats`	`5`	Elected seats that rotate through the standby bench.
`consensus.producer-candidate-pool-size`	`100`	Standby bench size for fair rotation.
`consensus.epoch-height`	`17000`	Main blocks per epoch (≈ 3 days; active set re-elected each epoch).
`consensus.time-slot-duration`	`12000 ms`	Active production window per slot.
`consensus.time-slot-interval`	`3000 ms`	Intermission between slots (full slot = 15 s block).
`consensus.amount-promoter-tx`	`2000`	Non-refundable registration deposit (→ treasury).
`consensus.fee-promoter-tx`	`3`	Exact fee for a validator-registration transaction.
`consensus.safety-floor-validators`	`4`	Anchors never phase out below this many healthy validators.
`consensus.anchor-demotions-per-epoch`	`1`	Privileged anchors shed one per epoch as independents prove themselves.
`consensus.validator-proven-epochs`	`2`	Consecutive epochs a non-anchor must produce to count as proven.
`consensus.equivocation-ban-threshold`	`2`	Distinct proven offenses → permanent ban.
`consensus.slash-reporter-bounty`	`5`	Bounty paid to a slashing reporter (from treasury).

> **Legacy "promoter" names.** The API and this guide say **validator**, but some internal identifiers keep the original word **promoter** — the same concept under a legacy name. These are unchanged: the config keys `consensus.promoters-count` / `consensus.fee-promoter-tx` / `consensus.amount-promoter-tx`, the transaction/state types `PromoterTx` / `ExitPromoterTx` / `PromoterState`, and the `promoter_*` database tables.

Database (PostgreSQL)

By default the node uses embedded H2 (`jdbc:h2:file:./data/janzeer;MODE=PostgreSQL`), stored under the bundle's `data/` directory — zero setup, fine for a watch-only node or a small validator. To use PostgreSQL, set (the bundle launcher picks these up):

```
export DB_URL=jdbc:postgresql://localhost:5432/janzeer
export DB_USERNAME=janzeer
export DB_PASSWORD=********
```

Both drivers ship with the node and the Flyway schema baseline is dialect-agnostic, so no code changes are needed.

`config.json` — node identity

`config.json` holds the node's identity (its location can be set with `--node.config-path`; the bundle launcher points at `data/config.json`). If the file is missing, the node auto-creates it on first boot — generating a fresh key and seed phrase — which is what happens on a fresh download:

```
{ "externalPort": 9199, "secret": "<node-signing-key>" }
```

- **`secret`** — the node's block-signing key; its public key is the node's **validator key** (`nodeKey` in `GET /api/v1/info`). A blank `secret` is replaced by a freshly generated one on first boot. The node signs blocks and votes only once that key is registered and in the active set; until then it is watch-only.
- **wallet** — the node's own wallet, derived alongside the identity.
- **`externalPort`** — the P2P port advertised to peers.

Treat `config.json` as a secret and back it up — it contains the signing key and is the node's identity.

Genesis: `genesis.json`, pin, overrides

A node must reproduce the chain's two bootstrap blocks bit for bit. A downloaded node does so from the public **`genesis.json`** shipped in its bundle (`--janzeer.genesis.file=./genesis.json`, plus `--janzeer.genesis.hash=<epoch-1 genesis hash>` as a pin — both set by the launcher). The file carries the anchor set, the final genesis balances, the ceremony signer's *public* key and the two pinned hashes + signatures; the node rebuilds the blocks, checks the hashes and verifies the signatures before touching its database, and refuses to start (`GENESIS MISMATCH`) if the file, the pin and the ledger disagree. Any node serves its genesis at `GET /api/v1/info/genesis` / `janzeer_getGenesis` — that is where the release kit fetches it from.

The genesis anchors themselves (and the internal dev network) boot from properties instead. Two knobs let you bootstrap a fresh chain or pre-fund a wallet for testing. **Both**

are for development/testnet only — never enable them in production, and they are relevant only when standing up your own network (§6), not when joining an existing one.

- **Dev-fund faucet** — set `JANZEER_DEV_FUND=0xaddress:amount` (Spring property `janzeer.dev-fund`) to seed one funded account into genesis, so you can test sending coins without earning them first. **Off by default. Never set in production.**
- **Genesis override** — genesis is config-driven: `janzeer.genesis.anchors`, `janzeer.genesis.signer-secret` (the ceremony key — anchors only; a downloaded node never has it), and the optional `janzeer.genesis.founder` / `janzeer.genesis.distribution` custody addresses let the node seed a new genesis **at runtime** without editing source. If `.founder` / `.distribution` are absent, those allocations fold back into the key-less treasury so nothing is minted beyond the fixed 100,000,000 cap.
- **Faucet (private test network only)** — `janzeer.faucet.secret` (private key of a funded account of that network, on ONE node) turns on `POST /api/v1/faucet` (`janzeer.faucet.amount`, default 10; `janzeer.faucet.cooldown-hours`, default 24, per address and per client IP). The node **refuses** a faucet whenever `consensus.network-id` is the mainnet id, whatever is configured. `GET /api/v1/info` → `faucet: true` tells wallets and the explorer to show the faucet page.

The committed anchor secrets and genesis defaults are **dev/testnet values** — rotate them for any real deployment.

4. Becoming a validator

Registration is **permissionless** — there is no vote, no approval, and no lock-up. You become a validator by running a node with a valid signing key (§2) and submitting one on-chain transaction from the non-custodial wallet; registration alone admits you. The end-to-end walkthrough (wallet, deposit, the **Validator** tab) is in §2, Steps 4–5. This section explains how the role behaves once you are registered.

1. **Register** from the wallet that will receive rewards. The registration transaction (`POST /api/v1/transactions/validators`, carrying your node's `validatorKey`) costs an exact **fee of 3** plus a **non-refundable deposit of 2000** (`amount-promoter-tx`, paid to the treasury bank). The `validatorKey` must be unique. The deposit plus one-seat-per-key is the Sybil guard — it is **not** returned when you exit.
2. **Enter the active set by registration order.** Selection is by **registration order**, not votes or rating: eligible validators are ranked by their registration time (`createDate`, longest-registered first). While the number of registered validators is at or below the seat cap, all are active; overflow rotates fairly. The active set is the still-privileged **anchors** plus the longest-registered elected validators, filled up to `promoters-count = 21`. Of those, `rotating-producer-seats = 5` elected seats rotate round-robin (fair rotation, registration-ordered) through a standby bench of up to `producer-candidate-pool-size = 100`.

3. **Exit gracefully** when you want to retire. Submit an exit-validator transaction (`POST /api/v1/transactions/exit-validators`, fee ≥ 0.01) with your `validatorKey`; you are removed from the active set and slot schedule at the next epoch boundary, so you stop being assigned slots immediately rather than degrading liveness until downtime ejection. The registration deposit stays gone.

> **Validator grant (from 2026-10-10).** The first 10 outside operators who run a reachable, synced mainnet node for 7 days receive the registration deposit (plus the fee) from the ecosystem wallet. Rules and application: https://janzeer.org/validators#grant.

Rewards

When your node produces a block, the producing wallet receives **100%** of the block reward — there is no delegation, no commission split, and no reward claiming. The reward transaction is system-generated (fee 0, nonce 0) and validated against the canonical value on every node. Of each transaction fee, **80% goes to the block producer's wallet** and **20% returns to the treasury bank** (`treasury-fee-share-bps = 2000`); fees are never burned.

Automatic anchor phase-out

Anchors start privileged (always in the set, downtime-immune) and shed that privilege **one per epoch** (`anchor-demotions-per-epoch = 1`) once there are enough **proven independent** validators — non-anchors that produced in every one of the last `validator-proven-epochs = 2` epochs — never dropping the healthy set below `safety-floor-validators = 4`. If independents later collapse below the floor, dormant anchors automatically re-engage. A demoted anchor is not removed; it keeps validating on merit like any registrant.

Downtime ejection

A validator that stays **silent on its slot** is treated as offline and is **ejected from the active set at the next epoch boundary** (epochs are `epoch-height = 17000` main blocks, ≈ 3 days at 15-second blocks). Keep your node up, synced, and clock-correct so you never miss a slot.

Slashing for equivocation — run exactly one node per key

Casting two conflicting votes for the same slot/stage is **equivocation** (a double-vote). It is provable: a conflicting second vote produces a `SlashingTx`, and `equivocation-ban-threshold = 2` distinct proven offenses results in a **permanent ban**. The reporter earns a bounty of `slash-reporter-bounty = 5` from the treasury.

The operational consequence is strict: **run one and only one node per signing key**, and **keep that node's clock synced**. Two nodes sharing a key, or a misbehaving/duplicated instance, can produce conflicting votes and get your key permanently banned. Never copy `config.json` to a second running node.

5. Monitoring & operations

Health and metrics (actuator)

Spring Boot Actuator endpoints are served **unwrapped** (not under the API envelope):

Endpoint	Purpose
`/actuator/health`	Liveness/readiness health check.
`/actuator/info`	Build/info details.
`/actuator/metrics`	Metric catalogue and values.
`/actuator/prometheus`	Prometheus scrape endpoint.

Point Prometheus at `/actuator/prometheus` for time-series monitoring and alerting (CPU, memory, JVM, peer counts, etc.). The public bundle's launcher exposes **only** `/actuator/health` by default (`--management.endpoints.web.exposure.include=health`); add `prometheus` to that list in the launcher's environment to scrape a downloaded node, and keep the actuator behind a firewall or reverse proxy. `GET /api/v1/info/hardwareInfo` is off unless `node.api.hardware-info=true`.

Chain & node status (REST API)

Endpoint	Purpose
`/api/v1/explorer/info`	Network overview: `nodesCount`, `blocksCount`, `transactionsCount`, `blockProductionTime`, `transactionsPerSecond`, `epochNumber`, `epochDate`, `validatorsCount`, `maxSupply`, `circulatingSupply`.
`/api/v1/info`	This node: `nodeKey`, `host`, `port`, `networkId`, `genesisHash`, `chainSpecDigest`, `version`, `apiVersion`, `protocolVersion`, `syncStatus`, `faucet`.
`/api/v1/info/genesis`	The public `genesis.json` of this chain (what a downloaded node boots from).
`/api/v1/info/version`	Node version.
`/api/v1/info/uptime`	Node uptime.
`/api/v1/validators/active`	The current active set (paged).

Use `/api/v1/explorer/info` to confirm blocks are advancing and `/api/v1/validators/active` to confirm your key is in the active set.

Logs

- Native/single node: stdout (and the H2 data under the bundle's `data/` directory).
- Internal multi-node network: `run/logs/node{1,2,3}.log`; tail them all with `./run/nodes.sh logs` (see §6).

The SYNCHRONIZED gate

Production is gated on the node's sync status. A node only produces and votes once it reaches **SYNCHRONIZED** status (and only while its clock is NTP-synced). After a restart or when joining the network, expect the node to sync first — it walks epochs backward to fill gaps, validates per epoch, and saves forward before it can produce. If your node is up but not producing, check that it is synchronized and clock-synced before anything else.

Backup

Back up, in order of importance:

1. **`data/config.json`** (`config.json` on the internal layout) — the signing key and the node's identity. Losing the key is recoverable (you can re-register, paying the deposit again), but a **leaked** key is dangerous; protect it like any private key.
2. **The database** — H2 files under `data/` (or your PostgreSQL database). State can be rebuilt by re-syncing from peers, but a backup speeds recovery.

Also keep your **wallet recovery phrase (seed)** backed up offline — it controls the wallet that receives your rewards and paid the deposit.

Upgrades

0. Linux/macOS: `./update.sh` does steps 1–4 for you, with signature verification (Step 5 above).
1. Download the new bundle for your platform and verify `SHA256SUMS`.
2. Stop the node.
3. Unpack the new bundle over the folder (or pull the new Docker image); **keep the `data/` directory** (identity + ledger) in place — for Docker it is the mounted volume.
4. Restart and confirm the node reaches SYNCHRONIZED and, for a validator, re-enters the active set (`/api/v1/validators/active`).

Upgrade validators one at a time so the network keeps a live quorum — unless the release notes say the wire protocol version changed, in which case every node must move to the new release (old and new releases refuse each other's frames).

6. How the node is distributed

The node is **closed source and ships only as native binaries** — there is no JAR, no JDK to install and nothing to build:

Platform	Archive	Start
Linux x86-64	`janzeer-node-<version>-<network>-linux-x64.tar.gz`	`./start.sh` (a `janzeer-node.service` systemd unit is included)
Linux arm64 (Raspberry Pi 3/4/5, 64-bit OS)	`janzeer-node-<version>-<network>-linux-arm64.tar.gz`	`./start.sh`
Windows x86-64	`janzeer-node-<version>-<network>-windows-x64.zip`	`start.cmd`
macOS (Apple silicon / Intel)	`janzeer-node-<version>-<network>-macos-{arm64,x64}.tar.gz`	`./start.sh`

Platforms are published as they are built — the **Validators** page on janzeer.org marks the ones that are not available yet as "coming soon", and lists the SHA-256 of every archive next to `SHA256SUMS`. A bundle contains the `janzeer` binary (the explorer and web wallet are inside it), the network's public `genesis.json`, the launcher and `README.txt` — **no keys and no `config.json`**: your node creates its own identity on first start (§2, step 3).

One bundle = one network: the network id and the genesis hash are pinned in the launcher, so a testnet bundle can never join the mainnet. To upgrade, unpack the new bundle next to the old one and keep your `data/` directory (§5 *Upgrades*). Release notes state the **P2P protocol version**; when it changes, nodes of the old release are refused by the network until they upgrade.

7. Production hardening & security checklist

- ☐ **Rotate all dev/testnet secrets.** The committed anchor secrets and genesis defaults are for dev/testnet only — generate fresh keys and a fresh genesis for any real deployment.
- ☐ **Dev-fund faucet OFF.** Ensure `janzeer.dev-fund` / `JANZEER_DEV_FUND` is unset in production.
- ☐ **One node per key.** Never run two nodes with the same `secret`, and never copy `config.json` to a second live instance — equivocation leads to a permanent ban. The public bundle ships no `config.json` on purpose: never reuse someone else's.
- ☐ **Clock sync.** Configure reliable `node.ntp-servers` and monitor NTP offset (`ntp-offset-threshold = 250 ms`); a node with a bad clock silently stops producing.
- ☐ **Never edit `consensus.*` on a live network.** These params are pinned to the genesis chain-spec digest and verified in the handshake; changing one forks your node off the network. Only touch operational settings (ports, DB, peers, NTP).
- ☐ **Maintain a live quorum.** Finality needs `floor(2N/3) + 1` of the active set; keep enough validators up (all three anchors for the default network, and ≥ 4 validators to

tolerate one fault) and upgrade one at a time.

- [ ] **Protect the keys.** Restrict file permissions on `config.json` (and `run/secrets.env` for the internal multi-node setup); keep them out of version control and out of logs/backups that others can read. Back up your wallet seed offline.
- [ ] **Firewall the ports.** A validator exposes its P2P port (`9199`, or `9190…9193` for the anchors); a watch-only node exposes nothing. Restrict or proxy the REST API (`7019`) and especially the `/actuator/**` endpoints — do not expose actuator publicly (the bundle exposes `health` only). Keep `node.api.hardware-info` off.
- [ ] **Faucet only on a testnet.** `janzeer.faucet.secret` is refused on the mainnet id anyway; never fund a faucet account with real value.
- [ ] **Use a managed PostgreSQL** for production (`DB_URL/DB_USERNAME/DB_PASSWORD`) with its own backups, rather than the embedded H2 default.
- [ ] **Persistent storage.** Keep the bundle's `data/` directory and `config.json` on durable storage (for Docker, mount a volume) so identity and state survive restarts.
- [ ] **Monitor & alert.** Scrape `/actuator/prometheus`, watch `/api/v1/explorer/info` (chain advancing) and `/api/v1/validators/active` (your key present), and alert on downtime so you are not ejected for missing slots.
- [ ] **Back up regularly.** Keep secure, current backups of `config.json`, the database, your wallet seed, and (for a custom genesis) the anchor seeds.

Document status — version 1.2 (public node bundle, genesis.json; mainnet is the only public network, no public faucet); reflects node version 0.1.14 / wire protocol 3.4.0. Exact byte-level facts (crypto, transaction/block layouts, consensus math) live in the Yellow Paper.