Blueprint
by Hivemind

RESEARCH REPORT

Efficiently Scaling Operations in Multi-Chain Staking

How Blueprint's three-primitive operating layer handles upgrades, migrations, and recovery to maximize performance yield across a growing fleet of validators.

MATT BRASSEY · ENGINEERING MANAGER, STAKING · NOVEMBER 2026

MB
Matt Brassey
Engineering Manager, Staking, Blueprint

NOVEMBER 2026 · 13 MIN READ

— Executive Summary

Most staking operations are organized by chain. One group learns Ethereum, another learns Solana, and a contractor picks up whatever launched last quarter. Each network ends up with its own scripts, its own runbook and its own particular way of failing at two in the morning.

That arrangement works at a handful of validators, but it gets more expensive and more fragile with every network added: another team, another set of scripts, another set of failure modes. The cost shows up in the moments that decide yield: an upgrade that restarts at the wrong time, a migration that leaves a validator dark for an hour, a recovery that waits on a full resync, or a signing key that ends up live on two machines at once.

At Blueprint, we built one operating layer instead. It has three standard operations, or primitives, called **prepare**, **upgrade** and **cutover**, and they handle every upgrade, migration and recovery the same way on every network. The layer is called **Mainspring**. Behind it sit 20+ protocol plugins, including Ethereum, Avalanche, Canton, Sui and Monad, and a new network inherits all three primitives the day its plugin lands, so the fleet can grow without the operation growing with it. Our two newest, Monad and Sei, went from a standing start to fully tested on mainnet in about seven hours.

This piece covers what the three primitives do, how a new client release becomes a running node without anyone typing a command on a server, where AI agents fit and where we deliberately keep them out, and what the results look like in practice, through the lens of our Solana validator compared to infrastructure run by the largest operators on the network. Solana is one example, chosen because its data is public.

By the Numbers

We measure the operation on what a move costs, how quickly a handover completes, how far the same layer reaches, and how the returns compare with those of large operators.

0	16.2s	20+	1st
missed attestations moving our 2,048 ETH validator to new hardware	signing handover, timed to land between the validator's duties	protocol plugins behind one interface, in the cloud or on bare metal	of 14 institutional Solana validators on net protocol APY, Oct. 2026

> Figures in this report come from our own infrastructure or from public datasets. Where a number can be checked on-chain or against a public dataset, the source is named so you can check it yourself.

1. Where Validators Actually Break

A validator in steady state is not very interesting. It follows the chain, signs what it is asked to sign and collects rewards. The risk comes when something about it has to change, and that happens in three situations: a node has to be brought up from nothing, it has to be upgraded, and it has to be moved or recovered when something goes wrong. We solve each with its own primitive.

1. Brought up from nothing: Prepare

New hardware, a new region, a new network. Done the usual way, the old validator stops, the new one syncs, and the delegator absorbs the gap. Prepare builds the new node while the live validator keeps signing.

2. Kept current: Upgrade

Clients ship constantly, and some releases carry hard network deadlines. A restart at the wrong moment misses duties, and a failure halfway can leave a node unable to start. Upgrade does the slow work first, then restarts between duties.

3. Moved or recovered: Cutover

Hardware dies and disks fill. Syncing a replacement from scratch can take days, and a careless move can leave two machines signing with one key, which the network slashes. Cutover hands the identity to a ready standby, one signer at a time.

Chain-by-chain operations handle each of these differently on every network, which means the riskiest moments in a validator's life are also the least practiced. You might upgrade a given chain a few times a year and move a validator once. Nobody gets good at something they do once a year, on a different chain each time, under a deadline.

So we stopped treating them as chain-specific events. Each became a **primitive**: a single, tested operation with the same shape, the same safety checks and the same command on every network. What changes from one network to the next is the plugin underneath, which describes the chain.

One Specification, Any Protocol

Mainspring is the software layer we built to operate validators. Engineers and automation give it one of three commands, and it carries out the work the same way on every network. A protocol joins by implementing a small, strict specification. The plugin answers a fixed set of questions about its network. Everything else, from provisioning, monitoring and key custody to rollback and cutover, comes from the shared layer.

What it runs

Where the client comes from, which versions are pinned, and what the machine needs to run it well.

How it gets ready

The fastest trustworthy path to the chain tip, and a plain-language definition of "ready" that the primitives enforce.

How it is judged

Health from the node, and participation from the network itself, so a node can't pass by reporting on itself.

Who it is

Where its signing identity lives and how that identity is backed up, restored and handed over.

How it stops signing

What it takes, on this network, to prove an old signer can never come back.

What to watch

The alerts and metrics that matter for this chain, wired into the same monitoring as every other.

// FLEET-AUTOMATION

— 2. Mainspring Runs the Fleet

Mainspring prepares, upgrades and cuts over our validators, and we control all of them through its commands. Each protocol is a plugin behind the same three primitives, so adding a chain, replacing a machine or recovering from a failure is the same audited operation every time, never another bespoke setup.

The steps Mainspring takes are identical across every plugin; only the plugin's answers change.

ethereum | lido-csm | avalanche | canton | audius | monad | sei | cardano | sui | near | + more

01 Prepare

Builds a standby at the chain tip on any machine, cloud or bare metal, while the live validator keeps signing. Sync time is never downtime.

02 Upgrade

Stages a release without restarting anything, then switches only what changed, between duties, behind health and network-participation gates.

03 Cutover RUNNING

Hands the signing identity to the standby in seconds, between duties. The old signer is fenced and verified first, so two machines never sign at once.

```
ops@blueprint:~/mainspring                                  cutover

# Ethereum plugin
$ mainspring cutover ethereum --to standby
  ▸ standby at tip · identity staged, encrypted                  ✓
  ▸ window: between duties                                       ✓
  ▸ source fenced · cannot restart · verified                ✓ 4.4s
  ▸ signing history carried and checked                          ✓
  ▸ standby signing                                         ✓ 16.2s
  ▸ network confirms participation                      ✓ 0 missed
  ● identity live on standby · one signer, never two
$
```

20+	3	0	0
PROTOCOL PLUGINS, ONE LAYER	PRIMITIVES RUN IT ALL	MISSED ATTESTATIONS, PRODUCTION CUTOVER	BESPOKE SETUPS

An illustrative session with simplified commands. The steps and outcomes shown are those each primitive enforces; the Ethereum cutover timings are from our production move on 21 October 2026.

Prepare: A Standby at the Chain Tip

The deploy primitive builds a validator that is ready to sign but isn't signing. One command takes an empty machine, in the cloud or on bare metal, to a synced mainnet node with networking, metrics and alerting in place. It returns only when the plugin's readiness check confirms the node is **at the tip of the chain**. On most of our networks that takes well under an hour from a blank machine.

More important than the time is where it happens. The node being prepared holds no live signing identity, so however long a sync takes, the production validator keeps validating throughout and none of that time is downtime. Where a public snapshot is the fastest path, the standby uses it. Where it isn't, **Mainspring** copies the chain data from one of our healthy nodes, so the new machine starts minutes behind the tip rather than days.

Two refusals are built in. Before launching anything, a check searches everywhere we operate for a live validator with the same identity and stops if it finds one. It cannot be overridden. And a machine that loses its local disks rebuilds itself on boot, but it will never generate a new identity to do so.

Upgrade: Prepare, Then Switch

An upgrade follows the same logic, with the slow work kept away from anything that could affect signing. **Prepare** downloads, stages and verifies the new release and restarts nothing. If it fails, the node is untouched and still validating. Only then does **switch** restart the components whose version actually changed. Where a restart could cost a duty, the switch waits for a window in which none of our keys is due to sign, and on Ethereum a client roll now costs no attestations.

After the switch the node has to prove itself twice: once to its own health check, and once to the network, through a participation check that reads what the chain says the validator did. A failure rolls back automatically, with one deliberate exception. Clients whose databases migrate forward are never downgraded, because putting an old binary on a migrated database is how a node ends up unable to start. We learned that one the hard way, and the rule now lives in code.

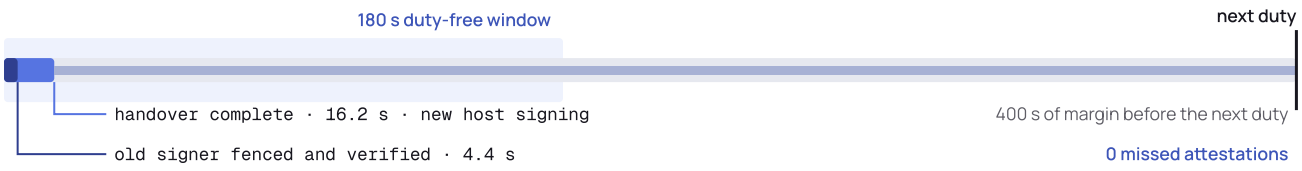
Cutover: Zero Downtime, One Signer

The cutover has the most at stake, so it is the step we engineered hardest. It moves a signing identity from the running validator to the prepared standby, and it rests on a single rule that every plugin implements for its own network: **the old signer must be provably unable to sign before the new one is allowed to.**

The old machine is locked out first, in a way that survives a reboot or an automated restart, and the lockout has to be verified before anything proceeds. The record of what the validator has already signed moves with it, so it can never sign the same thing twice. On networks with scheduled duties, the whole handover is timed to land in the gap after one duty and well before the next, so a duty is never missed.

Because the lockout is verified, the new machine can start signing immediately. Cautious setups often wait several epochs after a move, watching for a duplicate signer before they dare to sign. We don't need that wait, because the duplicate has already been ruled out.

PRODUCTION CUTOVER: 2,048 ETH VALIDATOR, ETHEREUM MAINNET, 21 OCTOBER 2026



Our 2,048 ETH validator moved to a new host and a new execution client in production. The signing handover took **16.2 seconds**, of which fencing the old signer took 4.4. The destination was ready **400 seconds before its next duty**, and the post-cutover check read the chain and found zero missed attestations.

The same pattern holds on every network, using whatever mechanism fits it, and every plugin that holds keys implements it. In October we moved all of our Avalanche validators onto new hardware the same way, one at a time. Each new machine started from a copy of a healthy node's chain data, caught up while the old validator kept running, and took over its identity in a single handover. Every old host stayed parked inside a rollback window before it was retired.

When Something Breaks

Recovery is just a cutover that nobody planned. Every signing identity is backed up, encrypted, the moment it exists, so losing a machine is a restore and never a loss of identity. When a host fails, its validator moves onto a standby that is already at the chain tip, or a healthy volume from a sibling node is carried into place behind it. Either way the recovery adds **no sync time** to the outage, and the one-signer rule holds throughout.

Access

Validators expose no inbound administrative access. Every operator action is authenticated, scoped to its task and logged, and interfaces that can sign or spend never face the internet.

Keys

Signing identities are encrypted with AWS KMS under a separate key for each protocol, at rest and while they move during a cutover. Some institutional Ethereum deployments also use remote signing, so the validator client never holds a key.

What the pipeline can touch

The upgrade pipeline runs on short-lived credentials trusted only from the reviewed main branch. It can roll a version. It cannot create or destroy a machine, change permissions or read a secret.

3. From Release to Running Node

Because every upgrade looks the same, the decision to upgrade can be separated from the act of upgrading, and the act can be automated end to end.

The source of truth is a version-controlled manifest that lists the version every deployment should be running. A watcher checks upstream releases several times a day. When a new version ships, it opens a pull request that changes only the version lines, and the description tells the reviewer exactly what they are signing off on: the scope, whether the change touches nothing, the node or the signing path, any network deadline with a countdown, and what merging will authorize.

Most releases need nothing more than that. The larger ones, such as consensus upgrades, database migrations and client changes that alter how a node has to be stopped or started, need real work alongside the version change: new configuration, a new safety check, sometimes a change to the primitive itself. That work is built first and run end to end on test machines in AWS against mainnet, with keys that aren't staked, before the pull request ever reaches an engineer. By the time someone is asked to merge, the upgrade has already been done once for real.

> **"Nothing rolls because a release exists. It rolls because a human merged."**
>
> From the **Mainspring** upgrade runbook

DECISION · HUMAN

01 Release detected
Upstream sources and the chains themselves, checked several times a day.

02 Pull request opened
Version lines, plus any config or primitive changes, already tested on AWS. Scope, impact and deadline stated.

03 Engineer merges
Review is the approval. Merging is the only trigger.

04 Safety floor
Refuses to run upgrade logic older than the newest reviewed version.

EXECUTION · DETERMINISTIC, ONE NODE AT A TIME

05 Baseline
Records what the network already says about the node, so the roll is judged only on what it changes.

06 Prepare
Stages only what changes. Restarts nothing.

07 Switch
Restarts changed components, inside a duty-free window where signing is at stake.

08 Settle, soak, verify
Health and network participation, a soak period, then the running version must match what was merged.

Fleets roll one node at a time and stop at the first failure. The pipeline refuses to run if it cannot confirm it holds the newest version of its own upgrade logic, which closes the gap where stale code rolls back something it no longer understands. Since early September, dozens of upgrades have shipped this way across our fleet.

For an engineer, keeping a fleet current comes down to reading a pull request and deciding whether to merge it.

Where the Agents Fit, and Where They Don't

At Blueprint, we spend a large share of our week directing AI agents. They write plugin code, run test suites, read telemetry and draft incident analyses. They are very good at it, and they are a large part of how we cover this many networks without adding a team for each one. But models aren't deterministic, and we don't want anything improvising on a validator.

So agents only reach a validator through the same primitives an engineer would use. Everything that touches a validator is static, reviewed code with a test suite of more than 1,600 cases behind it, and it behaves the same way whether an engineer or an agent invokes it. Our operations agent, Bluemax, lives in the team's chat and can see the whole estate, from sync lag and alerts to versions and logs. On a node it can run a short, fixed list of read-only diagnostics and nothing else, and it never holds a credential.

Adding a Network

Because the specification is fixed, adding a protocol is mostly a matter of describing it. **Monad** and **Sei** are the most recent examples. We started on both early on 20 October 2026, building them in parallel with agents working against the plugin specification. Monad's first mainnet-tested plugin merged about three and a half hours later, and both were finished, tested and merged roughly seven hours after the work began, through the same review and CI checks as any other change.

Neither was called ready until it had been through its whole lifecycle live on AWS, syncing from real mainnet peers:

- **Deploy from scratch:** a blank machine to a node at the chain tip and ready to validate, timed. Monad took under sixteen minutes, on both a cloud host and the enrolled-host path its bare-metal deployments will use. Sei took about thirty-five.
- **Rebuild:** local disks wiped, then the node brought itself back without creating a new identity.
- **Upgrade:** a real client release rolled through prepare and switch, then a deliberately broken release to prove the rollback.
- **Cutover:** a test identity moved between machines several times, including cloud to enrolled host and back for Monad, with the old host fenced and retired each time.
- **Refusals:** a destination behind the tip, an already-retired source and a source that would not stop cleanly were each refused before the key moved.

The test identities were never registered or staked, so none of this touched a live validator set. Neither network needed new operational tooling; both picked up the upgrade pipeline, monitoring and key custody on the first day.

Cloud and Bare Metal

A plugin can target cloud machines, machines we enroll ourselves, or both, and an enrolled machine, including bare metal in a colocation facility, gets the same three primitives as a cloud host. That matters because some networks are best served by dedicated hardware. Monad does not officially support cloud hosts, so its plugin is built for bare metal first.

The same layer also runs data infrastructure. On Canton we operate a full archive alongside our validator: an index of every Canton Coin transaction since the network's June 2024 genesis, continuously ingested and typically seconds behind the head of the chain. It powers CC Scan, our public Canton explorer, and the API underneath it.

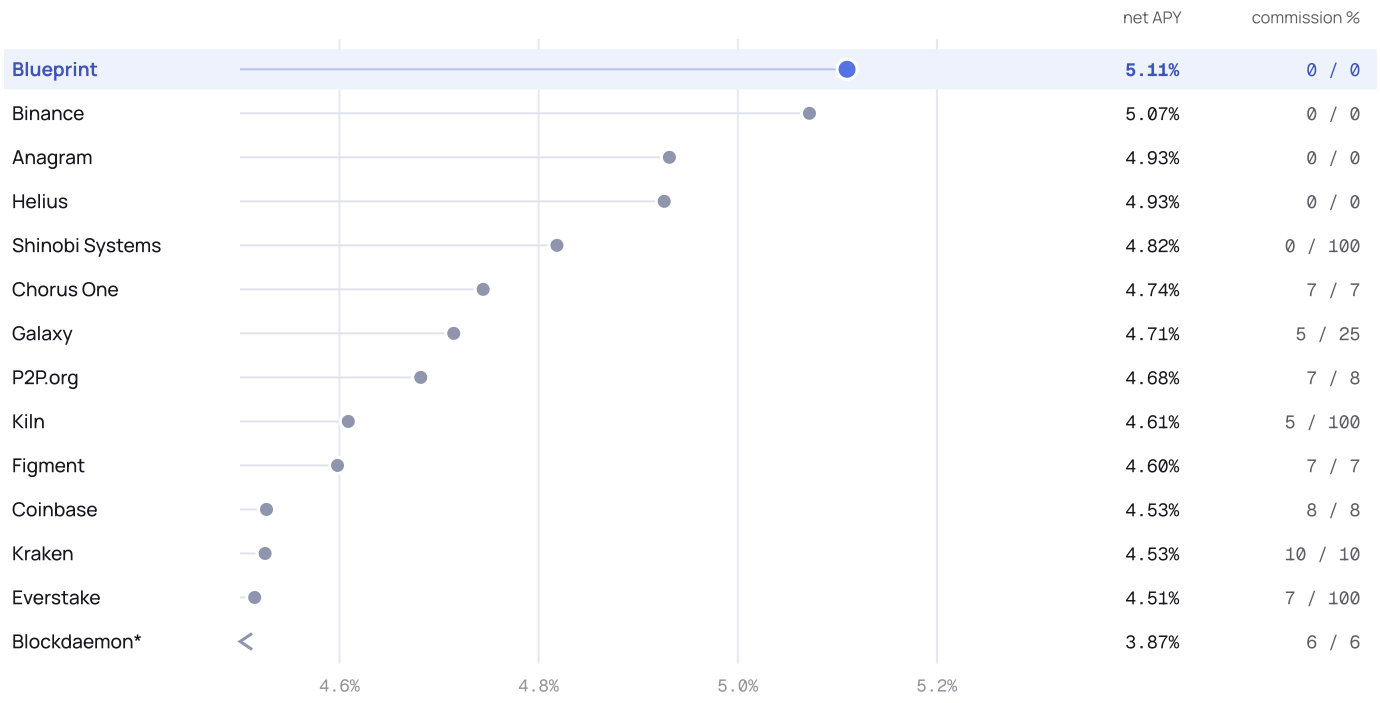
4. Results in Practice: Solana as an Example

We use Solana as the example here because its reward data is public, per-epoch and comparable across every validator on the network, which makes it the easiest place to check our work.

Our mainnet validator runs on dedicated bare metal in Frankfurt. It is connected to **DoubleZero**, a dedicated fiber network built for validator traffic, so block data moves over dedicated links instead of the public internet. It runs Jito's Block Assembly Marketplace (BAM) client and charges **0% commission** on both inflation rewards and MEV. It holds more than **600,000 SOL** of active stake, including a portion of the treasury of Nasdaq-listed Upexi, along with delegations from Jito, Shinobi Systems, The Vault, JPool, DoubleZero and Hivemind.

Every month we benchmark it within a fixed set of fourteen institutional validators, using per-epoch data from Trillium. The figures below cover October 2026, epochs 1046 to 1072.

NET PROTOCOL APY, OCTOBER 2026 (EPOCHS 1046 TO 1072)



Net protocol APY is inflation plus Jito MEV after commission, before any off-chain arrangement such as fee rebates or negotiated boosts. Same fixed vote accounts and the same time-weighted aggregation for every provider. *Off this chart's scale: Blockdaemon's benchmarked vote account shows no inflation rewards from epoch 1068 as its stake left. Source data: **Fueled by Trillium I Solana** (CC BY 4.0).

We finished **first of fourteen**, four basis points ahead of Binance and eighteen ahead of Anagram, and ahead of Helius, Galaxy, Coinbase, Kraken, Figment and the rest. Three numbers from the same period help explain the result.

15.9976

of 16 possible vote credits per slot, the highest average in the fourteen-validator set through epoch 1063, the last before Alpenglow. Credits are earned by voting quickly and correctly, and they translate directly into inflation rewards.

1.94×

the network median MEV per block. That is the median of per-epoch ratios, so it adjusts for slot-time changes and does not depend on one lucky epoch.

0 / 0

commission on inflation and on MEV. What the validator earns at the protocol level reaches the delegator.

Automation Around the Validator

The validator's revenue streams are claimed and moved automatically by small, single-purpose services that live apart from the validator itself. BAM subsidy claims run twice a day and look back far enough to stay inside the claim window. Reward tokens and excess SOL sweep to institutional custody every two days.

Alpenglow, Solana's new consensus protocol, went live on mainnet on 21 October 2026 at epoch 1064. Under it, every vote account pays a 0.8 SOL admission ticket each epoch, and a validator whose vote account runs dry stops voting. Our testnet validator, which runs release candidates ahead of mainnet, hit exactly that shortfall in September. By the time Alpenglow activated on mainnet, an automatic top-up was already watching the vote account. Its first refill ran that morning, hours before activation.

That is the usual order of things for us: the testnet runs into a problem first, we script the fix and put an alert on it, and mainnet never runs into it at all.

Commission (inflation / MEV)	Provider
0% / 0%	Blueprint, Anagram, Binance, Helius
0% / 100%	Shinobi Systems
5% / 25%, 5% / 100%	Galaxy, Kiln
6% to 8%	Blockdaemon, Chorus One, P2P.org, Figment, Everstake (7% / 100%), Coinbase
10% / 10%	Kraken

Commissions for the same fourteen vote accounts during October 2026, from the same Trillium dataset.

5. Built to Scale

Look again at the commission table. Part of what a commission pays for is the operator's org chart. If every network needs its own team, handoffs and runbooks, that overhead gets passed on, and delegators pay it every epoch.

Our costs don't grow that way. A new network is a plugin, a new node is one command, and keeping a fleet current is mostly code review. Engineering work goes into the shared layer once and every protocol picks it up. The duty-window timing we built for Ethereum upgrades now protects Lido CSM upgrades too, because both plugins describe their duties the same way.

The compartmentalized model

- Each chain is its own team, scripts and failure modes
- Hard operations are rare, so they stay unpracticed
- Headcount scales with the number of networks
- Overhead gets priced into commission

The Mainspring model

- Every chain runs through the same three primitives
- Every prepare, upgrade and cutover exercises the same code
- Headcount scales with the shared layer, not the chain count
- Less overhead to pass on to delegators

Going from dozens of validators to thousands doesn't change the design; the manifest just gets longer. The interface that rolls a fleet one node at a time today works the same at ten times the size, and a single control plane can keep every one of them current, monitored and accountable to what the network says it did.

People tend to assume careful means slow. The primitives wait where waiting matters (for a duty-free window, for proof the old signer is gone, for a soak to finish before an upgrade counts as done) and move quickly everywhere else. That is how the operation gets more careful and faster at the same time, however large the fleet becomes.

Most of the difference between good staking returns and mediocre ones comes down to upgrades, migrations and recovery. We built one way to handle all three, and nothing touches a **Mainspring** validator any other way.

Next: Monitoring and Alerting

Mainspring covers what we do to a validator. The other half of the operation is how we watch it: more than two hundred alert rules, health read from the network as well as from the node, and automated watchdogs that act when a client stalls. That system deserves its own piece, and we will cover it in a future post.

Talk to Blueprint

If you hold proof-of-stake assets on a treasury or fund balance sheet, or you are deciding who should run validators for a network you are building on, the most useful question to ask any operator is how they handle upgrades, migrations and recovery. We're happy to walk through our answers in detail.

- Stake with us: zero-fee staking across our supported networks through the Blueprint One membership.
- Dedicated infrastructure: white-label validator deployments on any Mainspring protocol, in the cloud or on bare metal, with zero-downtime cutovers built in.
- See the data: our Solana validator, vote account `528hi3StRe7uGjt99d35myh95JPc2MqBEHTPYcEhqMg5`, is public on every Solana explorer.
- Talk to us: theblueprint.xyz · sales@theblueprint.xyz

Blueprint Overview

Blueprint, a Hivemind ecosystem company, is building a unified operating system for digital assets. We offer institutional staking and validator operations across a broad set of networks, a portfolio management system with front- and back-office tooling, and data and development services. We work with asset managers, treasuries and foundations.

We spend our days where institutional capital meets chain infrastructure: custody integrations, validator operations, and reporting that has to stand up to an audit.

About the Author

Matt Brassey is Engineering Manager, Staking at Blueprint, where he designed and built **Mainspring** and the prepare, upgrade and cutover primitives behind it. He has spent seventeen years in infrastructure engineering, including six years running systems at IBM and two years in wallet operations at Bittrex, and now spends much of his time on agent-driven engineering for validator operations.

BLUEPRINT

Blueprint is the institutional orchestration platform for digital assets,
a Hivemind Digital Group ecosystem company.

Learn more at theblueprint.xyz