YOM

Decentralized GPU Edge Infrastructure

the fastest and most affordable GPU infrastructure on the planet

Technical documentation & whitepaper:
the network, the protocol, and how
operators earn.

V1.0 · 7 JUNE 2026 · YOM.NET/DOCS

Contents

BUILD

ABOUT

SECTION

problem

the post-cloud endgame

The narrative that we need *more* data centers is a lie.

The world is not short on computing power. It is drowning in misallocated computing power. The raw capacity needed to run AI, the metaverse, and the next generation of AAA gaming already exists. Most of it is sitting turned off in people's homes.

YOM is not building a better data center. We are building the layer that makes the data center optional.

The sleeping giant: 50-75% latent capacity

At any given second, 50% to 75% of the world's consumer GPU power sits idle.

Millions of RTX 3090s, 4080s, and high-end gaming rigs are running browser tabs or nothing at all. This is the single largest reservoir of compute on the planet, and centralized cloud providers treat it as if it does not exist. We treat it as the supply side.

Activating that hardware does not *add* capacity. It unlocks a resource that is already manufactured, already distributed, and already paid for.

And consumer GPUs are not the only supply side. Carrier-tower-sited GPUs through a Tier-1 telecom operator partner (scoped 200 towers near-term, scaling to 19,000 masts) add deterministic edge coverage where it matters. Hybrid topology, not a single-source marketplace.

The economic floor centralized cloud cannot cross

To scale, a hyperscaler has to spend. Doubling capacity means pouring billions into silicon, concrete, land, and cooling, and that mortgage sets a hard floor under their pricing. They cannot charge below roughly $2.00 per hour because someone has to pay for the building.

YOM has no building.

We do not buy hardware. We do not pour concrete. We orchestrate a protocol across hardware that already exists, which puts our blended cost near $0.13 per hour, about 15x below the centralized floor. That is not a discount. It is a different cost structure, and it is mathematically out of reach for anyone carrying datacenter capex.

Defeating the speed of light

Centralized cloud is also fighting physics, and losing.

You cannot optimize the speed of light. If the server is a thousand kilometers away, the latency is baked in, and no amount of fiber fixes the fact that the box is simply too far from the player. YOM solves latency by ignoring geography: every gaming PC on the network is a potential edge node, which moves the compute into the user's own neighborhood. Signal travels down the street instead of across the country, and the network holds sub-10ms median latency with nodes inside ~50km of the player.

That is local-console responsiveness, achieved through proximity rather than brute force.

What stays scarce when everything else goes to zero

The deeper shift is that software itself is collapsing in value. Anyone with a frontier model can generate their own app in a weekend. When everything can be generated, intelligence becomes abundant and software gets cheap. What stays scarce are the things you cannot prompt into existence: **latency, locality, energy, hardware, and physical presence.**

That is the ground YOM is built on. Owning information matters less when generation is cheap; operating low-latency systems close to the player matters more. The economy that follows rewards real contributors in real places, not distant monopolies. Miners, retail operators, and telcos contribute on the same open substrate. The network is permissionless.

The shift is structural

We are watching the transition from rent-seeking platforms to participatory networks.

- **Old world:** you pay a corporation to rent a computer far away. The corporation keeps the margin; the planet pays for new manufacturing.
- **New world:** you pay a peer for a computer nearby. The contributor keeps the reward; nothing new has to be built.

YOM is the infrastructure for that transition. We start with cloud gaming because gaming is the hardest real-time streaming problem there is: ultra-low latency, AAA fidelity, millions of concurrent sessions. Solve gaming and the same mesh serves AI inference, autonomous compute, and digital twins by inheritance.

And consumer cards reach further than the marketing assumes. Recent research on algorithmic communication packing has shown that the hyperscaler interconnect moat can be broken from the math side, not the hardware side. With matching software primitives in the stack, consumer GPUs can serve frontier-class training and large-model inference, not just inference at the edge.

Step one is gaming. Step two is everything else.

origins & vision

YOM was not born in a boardroom. It was forged in production. In 2020 the world locked down, and our founder Jorrit was running a high-end virtual production studio. To keep clients connected he delivered a massive digital twin of a TEDx stage, live motion-capture avatars streamed straight to thousands of browsers.

It worked, and it nearly killed the business. Running on AWS exposed a hard floor: despite extreme optimization, the internal cost stuck near $2.00 per user-hour. Centralized cloud was built for enterprise storage, not for mass-market interactive streaming. So we set out to solve a problem Amazon had no reason to fix.

The timeline

- **2021:** Secured a EUR 150K innovation loan to build a custom pixel-streaming backend. The MVP proved that a mesh of edge-located consumer GPUs could beat centralized latency, the insight the whole company is built on.
- **2022:** Concluded that only a community-owned distributed mesh scales cost-effectively to millions. Rohan Solanki joins to architect the HyperOrch scheduler and a custom NodeOS.
- **2023:** Won the European Commission's "Seal of Excellence" for deep-tech innovation. Shut down the agency to focus 100% on infrastructure. First go-to-market conversations with Andrew Pringle and Jeff Outlaw.
- **2024:** Formalized the C-suite with gaming veterans. HyperOrch alpha streams its first UE5 demo; 7,000+ node operators whitelisted.
- **2025:** Testnet live across 20+ regions. First NANO batch (400 devices) sells out in two days. Won "Best Gaming Tech Startup" at Gamescom. Migrated the network with 93.6% community retention.
- **2026:** 700,000+ registered users, six games live on testnet, 40+ studios signed. The production network now holds sub-10ms median latency, well past the original target.

A network owned by the people who run it

We are building toward an economy of real-time participation, not one of abstract credentials or centralized intelligence.

As the internet evolves from an **internet of files** to an **internet of events**, value moves with it. Intelligence becomes abundant; software gets cheap. What stays scarce are the things abstraction cannot erase: **latency, locality, energy, hardware, and physical presence.** In that world, power no longer belongs to whoever owns the information. It belongs to whoever can operate live systems close to human time, inside the hard limits of physics and perception.

YOM exists for a world where:

- **People are operators and owners** of the infrastructure that runs digital experience, not just users of it.
- **Value flows to the contributor, not the platform.** We replace dependence on distant monopolies with a direct economy that rewards people for the hardware, energy, and care they provide.
- **The contributor base is open.** Miners, retail operators, and telcos run nodes on the same substrate. The network is permissionless.
- **Digital reality stays grounded:** a system that has to be run, measured, verified, and paid for by real contributors in real places, not a spectacle to be consumed.

This is not theory. A single host PC running eight node licenses earns up to ~$466 per month streaming compute to nearby players (a single node earns ~$58), and operators net that after their own electricity in every major market. That is a new form of participation in the digital economy.

The mission

Turn locally owned, real-world hardware into active economic infrastructure.

We coordinate a global network of community-owned edge devices, orchestrated by our patent-pending HyperOrch scheduler and rewarded through revenue-sharing with operators, into infrastructure that gets cheaper, faster, and more distributed as it grows. The opposite of centralized cloud.

We start with cloud gaming because it is the hardest real-time streaming problem there is: ultra-low latency, AAA fidelity, millions of concurrent sessions. Pixel streaming lives natively on the edge, and locality is the entire game. Solve that, and the same mesh serves AI inference, autonomous vehicles, digital twins, and every other workload where centralized data centers hit their ceiling.

We exist to:

1. Keep **human agency economically valuable** in a world of automated cognition.
2. Convert **idle hardware into paid, verifiable productive capacity.**
3. Anchor digital economies to **locality and uptime**, not distant capital concentration.

Leadership

We combine deep-tech engineering with AAA gaming commercial experience.

- **Jorrit (CEO):** Pivoted from virtual production to edge infrastructure after hitting the limits of AWS.
- **Rohan Solanki (CTO):** Architect of the HyperOrch scheduler and NodeOS; expert in large-scale distributed systems and blockchain tooling.
- **Andrew Pringle (CCO):** Decades of retail, commercial, and distribution strategy.
- **Jeff Outlaw (CXO):** Gaming veteran connecting YOM to the global studio ecosystem and content pipelines.
- **Jeroen Elout (CMO):** Turns business requirements into beautiful UX and growth pipelines.

We are building the infrastructure of the future, grounded in human values.

SECTION

solution

how it works

Running a decentralized edge network for interactive workloads takes three things working together: a serious technical stack, on-chain economics, and an experience that feels effortless to players and developers.

YOM built an infrastructure protocol of patent-pending innovations to make streaming secure, efficient, and easy to use. The next chapter describes the architecture.

When you click “Play”

From a player’s point of view there is no download, no install, and usually no login. You click a link and within seconds you are playing a AAA game, rendered remotely and streamed to your browser or device. For players that means:

- Play instantly, even on low-spec hardware.
- No upfront cost (free-to-play or ad-supported sessions).
- Console-class responsiveness at sub-10ms median latency.

For studios and publishers, removing installs, patches, and hardware limits opens up far wider audiences and higher conversion.

What really happens

At a technical level, here is what happens when you hit “Play”:

1. The player requests a game via a web app, an embedded link, or a partner portal.
2. HyperOrch, our AI scheduler, finds the best nearby node based on latency, hardware fit, and current load.
3. That node, running securely on a NANO device, launches the game and streams it over WebRTC.
4. Gameplay streams peer-to-peer at sub-10ms median latency, as if it were running locally.
5. When the session ends, revenue is split programmatically on-chain:
    - 40% goes to the node operator
    - 55% funds the protocol treasury, operated by the YOM Foundation, which covers development, growth, and liquidity
    - 5% is burned, reducing token supply
6. Every transaction settles on-chain through smart contracts.

The YOM Foundation hosts the YOM DAO, the on-chain governance body that votes on protocol changes, treasury allocation, and incentive parameters.

A self-regulating economy

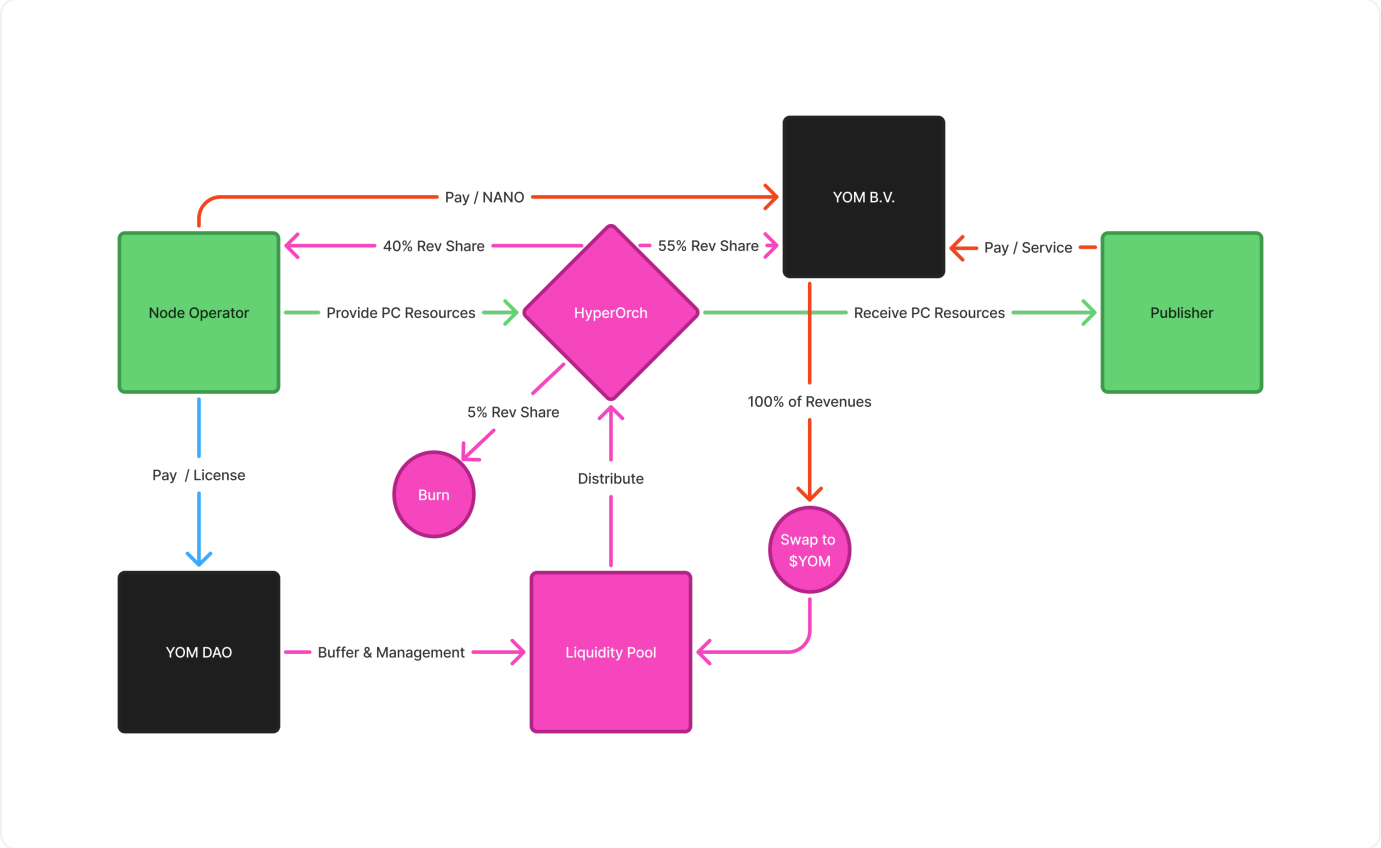
What keeps YOM balanced is a circular, self-regulating structure: players create demand, nodes serve it, publishers fund it, and every participant shares in the result.

Value ($YOM) flows according to a reputation system (XP), supply-and-demand pricing, and protocol-level splits: operator rewards, treasury, and burn. All of it is metered and priced on-chain.

Long-term governance runs through the YOM DAO, hosted by the YOM Foundation, built around the $YOM token and an XP-based node reputation system. Node-license proceeds fund the protocol treasury, which manages liquidity, governance programs, and ecosystem expansion. The result:

- High-performing nodes earn more and gain more influence over roadmap priorities.
- Protocol upgrades and incentive changes are community-led.

XP and $YOM are paid out automatically, per session and by performance, and bad actors are penalized without centralized enforcement.



For node operators

Operators contribute their own hardware using NANO, a plug-and-play secure-boot device that turns a standard gaming PC into a trusted, isolated node in minutes, with no technical hassle.

Operators benefit by:

- Earning token income by sharing idle GPU capacity, from ~$58 per month for a single node up to ~$466 per month for a fully-loaded eight-license host (see Maximizing Payout for the breakdown), net of their own electricity.
- Joining a community-owned economy where they control their hardware and their hours.
- Raising earnings through an XP-based reputation system (see Node Reputation).

Operators are free to join or leave at any time, which aligns supply with demand naturally.

For publishers & studios

YOM gives publishers a new channel for distributing and monetizing interactive content. Removing infrastructure and hardware barriers widens the audience and cuts distribution cost.

Publishers can:

- Deploy AAA games and immersive experiences straight into websites, apps, and social platforms in minutes, using YOM's SDKs.
- Cut infrastructure cost from the typical $1-2 per hour of centralized cloud gaming to per-session compute tiers from $0.02 to $0.15, roughly 15x cheaper at the unit level.

- Keep full control of monetization: premium, subscription, ad-supported, or free-to-play.
- Retain 100% of direct game-sales revenue.

With YOM, publishers reach new audiences and sustainable monetization without carrying datacenter cost.

blockchain

When a player presses **Play**, a node wakes up, a GPU spins, packets flow, and value moves, settled end-to-end on-chain. At the center of it is the **$YOM token**: the unit of account for compute, the reward for operators, and the supply that burns down with every session.

YOM settles on **Avalanche C-Chain** today, with cross-chain bridging to **Base** via LayerZero OFT. A single global $YOM token is used network-wide. This is a deliberate choice: at current volume, Avalanche C-Chain is more than sufficient, and a premature migration would only fragment liquidity. A sovereign L1 with regional tokens is on the roadmap (see below), not the current state.

Compute Coordination, Not General-Purpose Chain

$YOM is not a general-purpose smart-contract play. It is the settlement and coordination layer for a global network of GPUs. Every rendered frame and streamed session is a micro-economy of compute, bandwidth, and reward.

Under the hood, the **HyperOrch** scheduler assigns the optimal node by latency, capacity, and reputation. The node launches the game in a secure container, the session streams to the player, and the fee settles natively on-chain.

Rewards are tied to **sessions served, not blocks validated**, and priced in USD via fixed compute tiers (see Payout). This lets the network pay stable, predictable rewards without depending on the token's spot price. Operators contract with the protocol; the protocol distributes 40% to the operator, 55% to the Foundation, and burns 5%.

Because payout is earned by providing compute, $YOM is **not** staked for yield. Instead, holders can lock $YOM into liquidity pools on decentralized exchanges (including YOM's own) to earn farming rewards, and a soft-staking mechanism accrues XP that lifts a holder's standing from day one, securing the network and deepening trading liquidity at the same time.

Deflationary by Construction

The token has a strict deflationary structure: **no new tokens can ever be minted** (fixed supply). On every session, **5%** of the fee is burned, permanently removing supply. As usage grows, the float gets scarcer, aligning token holders with continued network activity rather than speculation.

Stewardship & Governance

The protocol and token are stewarded by the **YOM Foundation** (Seychelles, non-profit), structurally independent of the commercial entities that build on the network. The Foundation operates the treasury and contracts professional market makers for token liquidity.

Governance decentralizes over time. On-chain DAO voting on protocol parameters activates as the network matures (targeted 2027), with voting weight blending **XP reputation (~67%)** and **$YOM holdings (~33%)** so that the operators doing the work, not just the largest holders, steer the network.

Roadmap: Sovereign L1 & Regional Tokens

As transaction volume scales, YOM plans to migrate to a **sovereign L1** and introduce **regional tokens** (e.g. $yEU, $yCHN), each in a burn/mint relationship against $YOM, enabling local markets to price compute independently while $YOM captures value globally. The migration is gated on transaction volume reaching the point where gas on the

shared chain exceeds the threshold that justifies a dedicated L1, targeted **2027+**. Until then, all settlement remains on Avalanche C-Chain with a single global $YOM.

For the full token model, allocation, and TGE detail, see Token Allocation.

network

YOM is a decentralized GPU edge network: millions of idle consumer GPUs orchestrated into a global compute grid that delivers sub-10ms edge latency at a fraction of centralized cloud cost. Gaming is the first workload because it is the hardest, sub-10ms latency, AAA fidelity, millions of concurrent stateful sessions, but the architecture is a general-purpose substrate. The same network already serves enterprise interactive 3D and is built to serve low-latency AI inference next.

The stack spans a custom pixel-streaming solution, secure bare-metal edge computing, and intelligent decentralized orchestration. Together, these components stream any AAA game instantly to any device with sub-10ms edge latency, high reliability through rapid failover, and end-to-end security, all at a fraction of the cost of centralized cloud gaming.

The protocol is built on three components: the **HyperOrch** AI scheduler, **NodeOS** (the zero-trust streaming OS), and the **Universal Streamer** (browser-based pixel streaming). Nodes run on **bare metal, no Kubernetes and no virtual machines**, so each GPU is pushed to 100% for the game and context-switching latency spikes are eliminated.

Dual-Mode: Stateful and Stateless

Most distributed GPU networks operate stateless architectures: they route batch compute jobs to whatever GPU is free. Cloud gaming is fundamentally different. It requires stateful execution: game state persists across frames, save files survive session interruptions, and bidirectional input loops at sub-10ms latency.

YOM is both stateless and stateful by design. The same network serves stateful sessions (game streaming, interactive 3D) and stateless workloads (batch AI inference, ML jobs). The dual-mode architecture captures the cloud gaming opportunity that stateless-only networks structurally cannot, while remaining first in line for the low-latency AI inference demand forming on the same edge.

Peer-to-Peer Pixel Streaming

YOM deploys its own lightweight pixel-streaming service that runs games on remote consumer-grade GPUs and streams the video feed back to players via WebRTC for minimal latency. Once a session is approved, the gameplay is delivered over a direct peer-to-peer connection between the node and the player's device.

YOM's streaming protocol establishes an encrypted video/audio stream and input channel without routing through any central server, minimizing latency by eliminating extra hops. Game audio/video and player input data are end-to-end encrypted, so the content remains private and secure even if it traverses relays or public networks.

Universal Streamer & Middleware

Every YOM node runs a **middleware layer** that orchestrates the full game lifecycle: from pulling game images and spinning up GPU-accelerated containers, to managing tunnels, health checks, and session teardown. The middleware receives real-time instructions from the cloud backend and ensures each container is isolated, monitored, and ready to stream within seconds.

Inside each game container, the **Universal Streamer** takes over. This lightweight streaming client handles everything a player needs to interact with the game: WebRTC video and audio, low-latency input capture (mouse, keyboard, gamepad, microphone), and adaptive bitrate streaming, all running directly in the browser with zero plugins or downloads required.

The universal streamer is what makes YOM truly game-agnostic. Any game compiled to Linux x86_64 can stream on the network without modification. No SDK integration required. The streamer wraps the game's rendered output, establishes a peer connection to the player's browser, and starts streaming. For developers, this means onboarding a new title is as simple as uploading a build and embedding an iframe.

Together, the middleware and universal streamer form the operational backbone of every node, turning raw GPU power into seamless, browser-based gaming sessions at scale.

Discovery & Attestation

During live operations and at the end of every gaming session, the node reports telemetry (FPS, 95th percentile latency, thermals); if thresholds are exceeded it can dynamically scale down/up concurrent player streams of games, depending on the outcome of our scheduler.

Nodes publish signed availability records (location hash, GPU class, live load) to a *distributed hash table / on-chain registry*. Clients query the same ledger, pick candidates within a latency radius, then request the node's latest attestation. If the proof checks out the client proceeds; if not, the node is black-listed until it re-establishes trust. This serverless matchmaking eliminates single points of failure and allows the network to scale organically while remaining secure.

On shutdown, or if the boot drive is removed, the OS triggers a **rollback routine** that asserts a PCIe reset, DMA-scrubs RAM and VRAM, clears any drive keys, and reboots, ensuring no player data or studio IP persists. Updates follow an atomic A/B scheme: a new signed image is written to the inactive slot and only activated after signature verification and checksum pass, with automatic rollback on failure.

The nodes boot a minimal, read-only Linux image. On first launch the operating system:

- Auto-detects hardware & installs signed GPU drivers so the discrete card is exposed at full performance (no user intervention required).
- Runs a registration module that authenticates the user, measures uplink bandwidth, RTT to regional probes, idle GPU capacity, and temperature; it then publishes this capability vector to the discovery ledger for HyperOrch scheduling.
- Provides an isolated execution sandbox that mounts each game image read-only, passes the GPU through exclusively.

HyperOrch: AI Scheduler

Workload allocation across the YOM network is handled by HyperOrch, YOM's decentralized orchestration engine.

HyperOrch continuously gathers telemetry from each node, a multi-dimensional "capability vector" including available GPU/CPU capacity, memory, bandwidth, encoder load, thermal headroom, and more. Similarly, each game title is characterized by a resource footprint profile (GPU demand, memory needs, expected bitrate, etc.).

Using these inputs, HyperOrch's AI-driven scheduler predicts the expected performance (e.g. framerate) for a given game on each candidate node and dynamically matches players to the optimal node in real time. Unlike naive approaches that just pick the nearest server, HyperOrch filters and ranks nodes by multiple criteria (player-to-node latency, hardware suitability for the game, current load, etc.) to find a host that can deliver smooth 60+ FPS and sub-10ms ping for that session.

The orchestrator even learns from each session: it updates its model with actual performance outcomes, continuously improving placement decisions over time. This intelligent scheduling maximizes network-wide performance and efficiency, avoiding bad host-game pairings that might lag, fully utilizing available GPU headroom, and reducing costs by optimally balancing the load on community nodes.

In practical terms, HyperOrch's real-time decision-making ensures each player is connected to a node that will give them a console-quality experience, and it can proactively migrate or redistribute sessions if a better match becomes available.

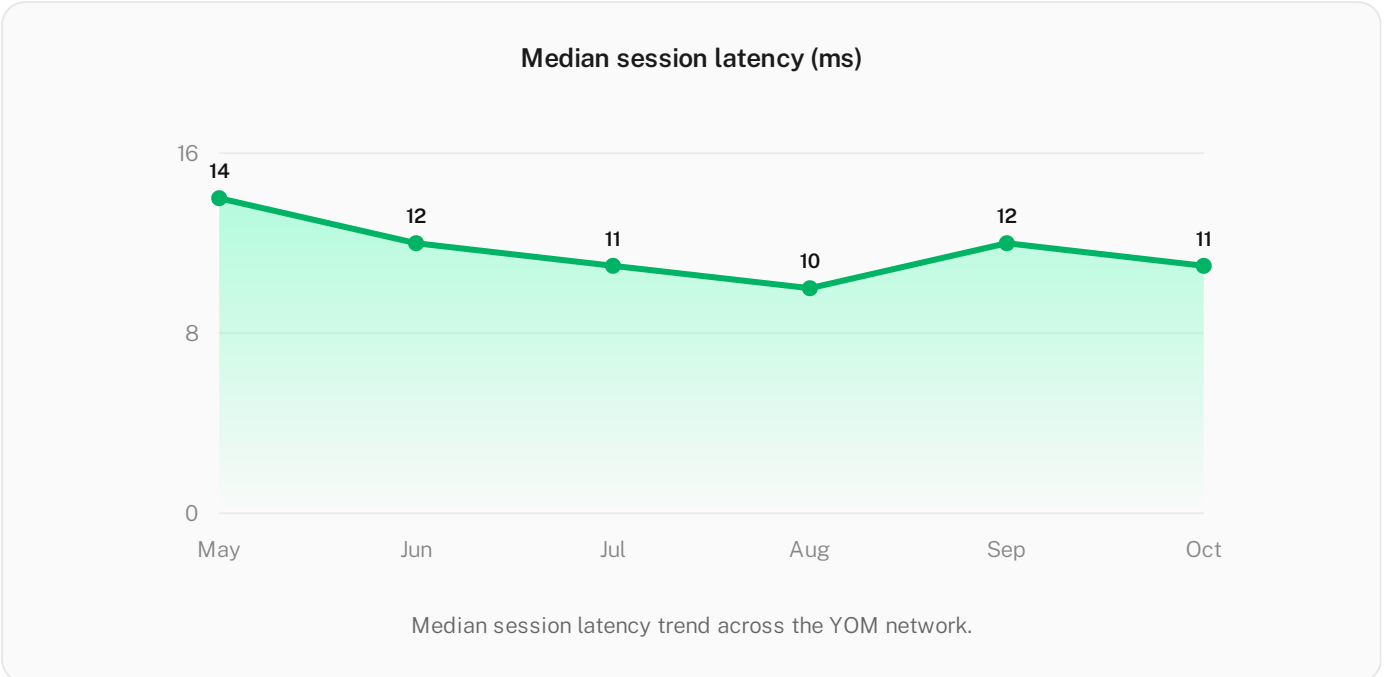
Cloud Backstop & SLA

The network's primary supply is community-operated consumer GPUs, but density takes time to build in any new region. In regions with insufficient node density, cloud GPUs provide a backstop that holds the service level while community supply catches up. This is an initial **5-15% of compute**, trending toward under 2% at scale, and it underpins a **99.5% uptime SLA**. Telco-hosted enterprise GPUs and hyperscaler burst capacity fill the gap; community supply takes over as it densifies. The result is a hybrid network that guarantees reliability from day one without waiting for full decentralization.

Patent-Pending Technology

YOM's core protocol is protected by three patents pending in Luxembourg, with US filings in preparation, and a fourth innovation in active R&D:

- **HyperOrch AI Scheduler** (LU603516): neural-net-driven session placement using multi-dimensional telemetry.
- **NANO Secure Boot Medium** (LU603514): a minimal, read-only Linux OS with a secure boot chain, so publisher IP is never exposed to the host machine.
- **Decentralised Pixel Streaming** (LU603515): game-agnostic WebRTC pixel streaming with adaptive bitrate, so any Linux x86_64 game streams without modification.
- **Vulkan VRAM Budgeting Kernel** (in R&D): a custom Vulkan-layer kernel that caps per-container VRAM usage on consumer GPUs, the software primitive that lets retail GPUs participate in workloads that traditionally required enterprise-grade isolation.



Median session latency trend across the YOM network.

operators

In the YOM ecosystem, **nodes** are the backbone of a decentralized GPU edge network. Each node is an operator-owned GPU contributing to the substrate that streams live 3D game sessions today and is built to serve enterprise interactive 3D and low-latency AI inference next. Unlike traditional crypto “nodes” that often sit idle or rely on speculative token emissions, our nodes perform real work and get paid based on actual usage.

What is a Node?

A node is essentially a secure container inside a gaming computer or server dedicated to running a single session for a player. When a player launches a game on the network, the game is rendered on a node operator’s hardware and streamed to the player’s device in real time. Each node in the network streams a session to a single concurrent player.

In simpler terms, running a YOM node is akin to hosting a small game server that *one player connects to for a session*, except the player only sees the video feed. Because YOM is decentralized, these nodes can be run by everyday operators on their own hardware. Every time content is streamed or rendered by your node, you get paid for that session.

Anyone can run a node, provided they obtain a node license and/or have access to suitable hardware or cloud infrastructure. A Node License is essentially an access token that gives you the right to stream more nodes from your host pc or the right to delegate it to other host. Every NANO Node Runner comes with a free embedded license by default. Subsequent licenses are digital NFTs. Once you have such a license, you can choose how to deploy your node: either assign it to your own PC/rig or delegate it to a verified Node-as-a-Service provider who runs the node on your behalf.

Restrictions on the emission of devices and NFTs allow the protocol to scale with the demand of the ecosystem. In practice that means we emit new Node Runners and Node Licenses to the market as we scale beyond the 60% utilization threshold.

How Many Nodes per Host?

Notably, one physical machine can run **multiple nodes** (consuming multiple licenses) in parallel if it has enough resources and bandwidth available. A single host can run **up to 8 nodes** by stacking 1 NANO Node Runner + 7 Node Licenses simultaneously on the same rig.

Nodes run on **bare metal, no virtual machines**. YOM does not slice the GPU into VMs; each node runs as a secure, isolated container with the GPU pushed to 100% for the game, which is what eliminates the context-switching latency spikes that a hypervisor would introduce. An operator with a high-end system multiplies their earnings by serving several players at once from a single machine.

Operator Earnings

A single node earns **~$58/month** at moderate utilization. Stacking up to **8 nodes on one host** earns up to **~$466/month** fully loaded. A NANO Node Runner is **$349** with one license included; each additional Node License is **$249**, putting payback at roughly **6 months**.

Configuration	Monthly Earnings	Investment	Payback
1 node	~$58/month	$349	~6 months
8 nodes (full host)	~$466/month	$349 + 7x $249	~6 months

Earnings scale with actual usage and node reputation. Node licenses are NFTs, so they are transferable on secondary markets and delegatable to a Node-as-a-Service provider who runs the hardware on your behalf.

Node Reputation

The reputation of a node is defined by various factors and is represented by a soul-bound (non-tradeable) metric called XP. The more XP a node operator acquired, the higher their position on the operator leaderboard. The higher your position on the leaderboard the more workload gets assigned to your node, meaning the higher your node utilization rate. Additionally high XP can also translate into winning prizes, badges, perks or bonus multipliers for node earnings. Core earning avenues are:

1. **Network Operation.** Node owners who run stable, high-uptime machines accumulate XP in proportion to the reliability and performance of their nodes. This ensures that those contributing critical infrastructure earn corresponding reputation points. And as a bonus simply by holding your rewards in your wallet, your nodes can accrue more XP.
2. **Community Participation.** Active Discord or forum participation, creation of high-quality guides or tutorials, and social media advocacy all generate XP. This mechanism rewards both the technical and cultural contributors who help grow and educate the community. The Web App and Discord facilitate easy access for members to participate in activities where they can earn XP.
3. **Community Jobs.** Periodic events hosted by the YOM Foundation or community-driven committees can offer bonus XP for tasks like stress-testing new features or helping onboard new members. Certain XP awards are distributed through peer recognition, where outstanding contributions or helpful community moderation receive upvotes or endorsements from other members.

payout

YOM's payout model is session-based and usage-driven, aligning rewards with real player activity. Nodes earn in proportion to the sessions they host, not by farming idle tokens. This chapter covers how sessions are priced and how rewards are calculated and distributed.

Compute Tiers

Every session settles at a fixed per-session fee set by workload tier. The tier is the workload-difficulty signal: heavier titles (higher resolution, framerate, and GPU load) settle at a higher tier.

Tier	Workload	Fee per session
Light	Mobile, 2D	$0.02
Standard	Indie 3D	$0.05
Performance	UE5	$0.10
Ultra	AAA 4K/60fps	$0.15

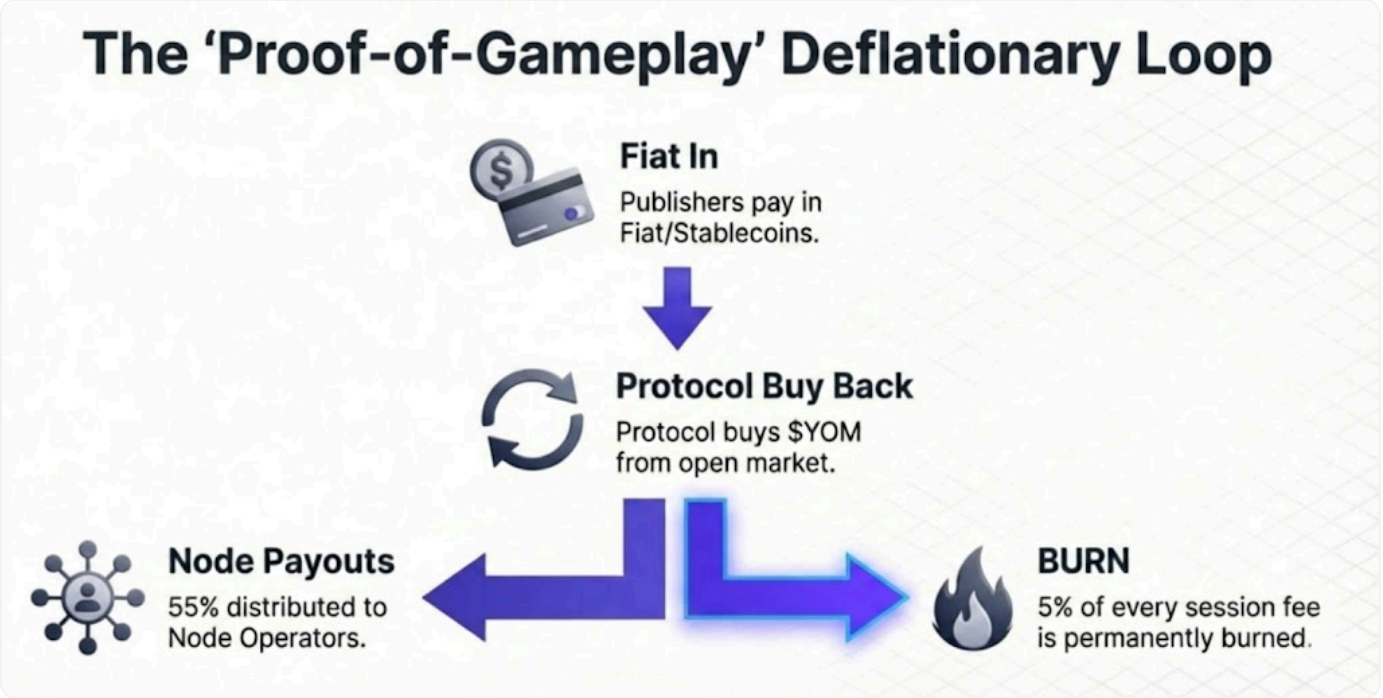
This blends to **~$0.13/hr** of streamed compute, an order of magnitude below the ~$2/hr floor of centralized cloud. The network collects telemetry from every session to assign the right tier.

Flow of Value

Every session settled through the YOM protocol pays the per-tier fee, and the protocol distributes it on-chain instantly. Operators contract with the **protocol**, not with individual publishers. The **YOM Foundation** is the counterparty on reward distribution, which is what keeps payouts stable and predictable regardless of which commercial entity collected the customer's payment.

Stakeholder	Split	Description
Node operators	40%	Paid directly to the operators who served the session.
YOM Foundation	55%	Funds protocol operations, treasury, and ecosystem incentives.
Burn	5%	Permanently removed from supply, keeping $YOM deflationary.

This 40 / 55 / 5 split is the protocol's economic invariant. It applies uniformly: Walter sessions, enterprise sessions, and any third-party aggregator settling through the network all pay the same split, on the same tiers.



Operators receiving rewards can hold to accumulate XP, provide liquidity on decentralized exchanges to earn farming rewards, or convert to cover costs. At moderate utilization a single node earns **~$58/month**; a fully loaded 8-license host earns up to **~$466/month**.

Pricing, Pegging & the Road to Regional Tokens

Session fees are priced in USD and settled in **$YOM** on Avalanche C-Chain today, with a single global token. This keeps operator rewards stable and predictable without depending on the token's spot price.

Regional price discovery is on the roadmap, not live today. As transaction volume grows, the network plans to introduce **regional tokens** (e.g. $yEU, $yCHN) on a sovereign L1, each in a burn/mint relationship against $YOM, so that local markets can price compute independently while $YOM accrues value globally. That migration is gated on transaction volume reaching the threshold where a dedicated L1 outweighs the liquidity cost of leaving Avalanche (targeted 2027+). Until then, settlement stays on Avalanche C-Chain in a single global $YOM. See Tokenomics for the chain and token detail.

In short: **your node earns by serving sessions, priced by workload tier, settled instantly on-chain**, with a uniform, predictable split that does not depend on who sold the session or where the player is.

SECTION

earn

nano

Our Plug-and-Play **NANO** is a node runner that offers a straightforward way to self-host a node, combining the benefits of full control with an easy setup process. With this option, you can quickly deploy your node without dealing with complex software installations. The NANO is **$349** and comes with **1 node license included**. It is manufactured and distributed via our hardware partner **SIMMS**, and sold through authorized resellers.

The first NANO Genesis batch sold out fast: **400 units in 2 days**. If you would rather not use a dedicated device, a **software-only node setup** is also supported on qualifying rigs.

Step 1: Rig Check

To run the NANO efficiently, your gaming rig must meet the following minimum hardware requirements:

Minimal System Requirements:

- **GPU:** Any NVIDIA RTX config (2060RTX and higher) with high VRAM to maximize node capacity.
- **CPU:** Intel i5/i7 or AMD Ryzen 7/9 for better multitasking and stability.
- **RAM:** 16GB would be sufficient on low config setups, but 32GB+ is recommended.
- **Internet:** Wired Ethernet connection (min 25mbps upload).
- **Cooling System:** Proper airflow and cooling solutions to prevent overheating.

You can run our Rig Check tool to understand your system's eligibility.

Step 2: Order the NANO

Once you've confirmed your system meets the requirements, you can order the **NANO** for **$349** (indicative price; regional taxes may apply). The device is built and distributed via our hardware partner **SIMMS** and sold through authorized reseller webshops such as Freshminers.com.

Step 3: Setting Up Your Node

Once your device arrives, follow these simple steps to get it up and running:

1. **Connect the NANO to Your PC.**
2. **Boot from the NANO.**
   - Restart your PC and access the BIOS by pressing `F2`, `F12`, or `DEL` during startup.
   - Set the YOM SSD as the primary boot device.
   - Save and exit the BIOS settings to continue.
3. **Follow On-Screen Setup Instructions:**
   - Once the system boots into the YOM Node OS, follow the guided setup to claim the device inside of your wallet.
   - After its connected you can access the system from the web portal by logging in with the wallet you provided.

That's it and you are now ready to rock & roll!

node licenses

After you successfully set up your NANO, you can multiply your earnings potential by adding node licenses to your setup. The maximum number of licenses on a single host is **8**. The actual number you can run depends on your bandwidth and your gaming rig performance.

Every NANO comes with **1 license included**. Each additional license is **$249**. The more powerful your gaming rig, the more licenses you can run. Each license requires about 25 mbps of upload bandwidth. You can check how many licenses you can run here.

Licenses are NFTs: they live in your wallet, are transferable on secondary markets, and can be delegated to a NaaS provider if you prefer not to run the hardware yourself.

From the node management view in the webapp, you can easily assign your acquired node licenses to your NANO. Alternatively you may want to decide to delegate your license to a NaaS provider to earn $YOM without running the hardware yourself.

How to get a Node License?

You can buy additional licenses at **$249** each from any supported marketplace or via app.yom.net/node-sales. Store them in the same wallet you connected your NANO to, so you can easily upgrade your device with more streaming capacity. Do note: licenses are in limited supply. To keep conditions healthy for all node operators, we unlock licenses and NANOs to the public as we pass the 60% utilization threshold.

naas delegation

License delegation goes live alongside our first major telco-tower deployment (targeted Q2-Q3 2026); it is not active yet. Node Licenses are NFTs, so they are transferable and delegatable: once live, enterprise GPU capacity from telco-tower and data-center partners will backstop under-served regions as the community supply scales.

If you don't own a gaming PC or don't want to use it for the network, you can delegate your Node License to one of our NaaS partners. Delegating your node to a **Node-as-a-Service (NaaS)** provider is the easiest, most convenient way to participate in the YOM network and earn passive income.

With delegation, you don't need to worry about your setup, monitoring, and maintenance. Instead, a trusted NaaS provider takes care of everything while you enjoy consistent earnings from your node.

Why Choose Delegation?

You pay a monthly fee to pay a NaaS service to host for you. In return you receive $YOM tokens for you value add to the network. Do note the number of tokens you received is entirely dependent on the performance of the NaaS service. Hence, while its the easiest to get started, it's not risk free. Many of our NaaS partners will attempt to deploy resources in area's with high demand. Others will not. Also their pricing varies. This is healthy competition.

How Delegation Works

After obtaining your Node License, follow these steps to delegate your node to a NaaS provider and start earning:

1. Navigate to **app.yom.net** and connect your Web3 wallet that holds the Node NFT.
2. **Navigate to the Node Management Section:**
   - In the dashboard, find the **"Node Management"** tab.
   - Select the node license you wish to delegate.
3. **Choose a NaaS Provider:**
   - Browse through the list of available Node-as-a-Service providers.
   - Evaluate providers based on their uptime guarantees, historical performance, and service fees.
4. **Confirm Delegation:**
   - Click the **"Delegate"** button and confirm the transaction in your Web3 wallet.
   - Once confirmed, your node will be handed over to the NaaS provider for management.

Running Delegated Nodes

Once your node is successfully delegated, you can monitor its performance through the YOM dashboard. It works in a similar fashion as the NANO dashboard, however with access to limited views. With delegation you can expect rewards that are usually somewhat higher than running on a NANO, however with the disadvantage that you have to pay a monthly fee in USDC.

Similar to the NANO, **payouts are processed automatically** at the end of each session and can be claimed directly from your Web3 wallet. Take into consideration; e**arnings vary based on: t**he overall demand for cloud gaming sessions; regional network saturation and available workload; your accumulated XP score, which influences workload priority.

Frequently Asked Questions (FAQs)

1. **Why does YOM use NaaS partners?** YOM makes use of NaaS partners on top of our DePIN to add a level of flexibility / scaling in our network, which only data centers can cater to. It ensures that we have more machines available in regions where we need to cater to high peak loads.
2. **Can I switch to self-hosting later if I change my mind?** Yes, you can revoke delegation at any time and transition to self-hosting if you prefer a more hands-on approach.
3. **What happens if my NaaS provider underperforms?** If your provider fails to meet uptime and performance standards and the amount of $YOM earned is beneath expectations, we recommend re-delegate your node to another provider or use the license for a self-hosting setup.
4. **Are there any fees associated with delegation?** Yes, NaaS providers charge a fee for the compute your rent from them. You pay this on a per month basis. Be sure to check provider details before delegation.
5. **How secure is the delegation process?** The delegation process is secured entirely on-chain, ensuring transparency and immutability. Your Node license remains safely stored in your wallet.

maximizing payout

Understanding how the network handles workload is crucial, because it directly affects your earnings. Every session settles through the protocol at a uniform split (40% to operators, 55% to the YOM Foundation, 5% burned), so your payout does not depend on who sold the session or where the player is. What you can control is how much of that flow you capture, and this guide is about optimizing exactly that.

Increase Node Count

The easiest way to increase your total earnings is to raise the maximum potential of your host. To do this, add additional node licenses as your hardware comfortably allows, maximizing GPU usage and total earnings. A single host can run up to **8 licenses**, and a fully loaded 8-license host earns up to **~$466/month** at moderate utilization.

Increase Node Utilization

Then you may also want to optimize your utilization rate of your nodes. The utilization rate is determined by your accumulated reputation (XP), which can be increased high uptime, reliable connectivity and community participation. Participate in community events and hold earned tokens to boost your node's reputation, placing you at the front of the line for session allocation.

The other nudge you can tweak is having your pc operate during times or days when there is high demand. Usually high gaming demand peaks evenings and weekends. Ensuring your node is available during these periods increases your likelihood of receiving session assignments.

Finally, prevent abrupt disconnects. Utilize reliable hardware, power backups, and monitoring tools to maintain consistent availability. If you go under once, that's acceptable. However, if that happens repeatedly, your host will get penalized when under a certain uptime rate.

Apart from the above factors, YOM's orchestrator also performs some balancing to prevent any single node from hogging all jobs or remaining completely idle for too long.

To Operate or Delegate

You can further optimize your rewards by ensuring the right balance of assigning your Node Licenses to your own host machine vs. delegating them to external providers.

Feature	Delegation	NANO
Technical Knowledge	None	Minimal
Setup Time	Very Low	Moderate
Optimization Potential	Can be optimized by picking the right NaaS.	Can be optimized by building your reputation.
Hardware	None	Gaming PC
Maintenance	You pay monthly service fees to the NaaS partner.	You pay for your own electricity bills.
Risk	Requires monthly commitment.	None, just turn off your PC at any time.

Earnings Expectations: Practical Examples

One of the most common questions potential node operators ask is "**How much can I earn?**". YOM provides a realistic forecast by modeling node earnings as a **Gaussian (bell curve) distribution** centered around an expected average.

The assumption is that on **average, a node will be utilized about 50% of the time** it's available. In other words, roughly half the hours your node is up, it will be actively streaming a session (the other half it might be idle, waiting for demand). Based on network simulations under current conditions, YOM estimates monthly earnings of **~$58 per node** at moderate utilization.

It's worth noting that **these numbers scale with the number of node licenses you run on your hardware** (assuming your machine can handle it). A single host can run up to **8 licenses**. For example, a fully loaded 8-license host could yield up to **~$466 per month** on average. With a NANO at **$349** (one license included) and additional licenses at **$249** each, the estimated payback on a fully loaded host is around **6 months**.

Illustrative 12-month operator ROI for a YOM NANO at default assumptions (RTX 4090, 24h/day). Actual earnings vary with hardware, uptime, and network demand.

referrals

Referral Program

Earn rewards by sharing YOM with your network. When someone purchases a YOM NANO (**$349**, one node license included) using your referral code, they save money and you earn commission.

How It Works

1. **Get your referral code:** Find your unique code in the Referrals dashboard
2. **Share it:** Send your link to friends, post on social media, or share in communities
3. **Earn rewards:** When someone makes a purchase using your code, you both benefit

Rewards

Benefit	Description
Discount for buyer	Percentage off product price
Commission for referrer	Percentage of product price (ex. VAT)

Your referral rate may vary. Check your Referrals dashboard for your current discount and commission percentages.

How Commission Is Calculated

Commission is calculated on the product price excluding VAT. The base NANO price is **$349** ex-tax; the worked example below shows the EU equivalent with 21% VAT applied.

Example: A buyer purchases a YOM NANO using a referral code with a 10% rate.

Step	Calculation	Amount
Retail price (incl. 21% VAT)		€369.00
Product price (excl. VAT)	€369.00 ÷ 1.21	€305.00
Buyer discount (10%)	€369.00 × 10%	-€36.90
Buyer pays	€369.00 -€36.90 + shipping	€350.25
Referrer earns (10%)	**€305.00 × 10%**	**€30.50**

The commission is based on the ex-VAT product price because VAT is collected on behalf of tax authorities and is not part of the product revenue.

When Rewards Are Confirmed

Your commission becomes confirmed once both conditions are met:

- The 14-day return period has passed
- The purchased NANO has been activated

You can track pending and confirmed rewards in your Referrals dashboard.

Payouts

Commission earnings are paid out at the end of each month to your connected wallet.

Sharing Your Code

You can share your referral link directly or use the quick-share buttons for X (Twitter) and Telegram in the dashboard. Your referral link format:

Leaderboard

Top referrers are displayed on the leaderboard. Climb the ranks by bringing more node operators into the YOM network.

SECTION

build

overview

YOM streams any Linux-compiled game to any browser. No app stores, no downloads, no device requirements. Players click a link and play.

What YOM handles for you:

- WebRTC video/audio streaming (H.264, VP8, VP9, Opus)
- Input capture: mouse, keyboard, gamepad, microphone
- Automatic scaling across devices and screen sizes
- Session management, matchmaking, and node allocation
- Analytics via Google Tag Manager integration

Four Ways to Build on YOM

Choose the path that fits your needs:

Path	What You Need	Time to Live	SDK Required?
Quick Start	Compile game to Linux, upload, embed iframe	~15 minutes	No
Web Integration	Custom UI, multi-stream pages, web-to-game commands	~1 hour	JS embed script
UE5 SDK	Responsive HUDs, game-to-web events, multiplayer, wallet features	1 to 3 days	Yes (optional)
Whitelabel Infrastructure	A branded surface to route players to: telco portal, OTT / smart-TV launcher, publisher launcher	Partner integration	Talk to us

Quick Start: Stream Any Game

Upload your compiled Linux binary and embed a single iframe. YOM's universal streamer handles everything else: WebRTC, input, video, audio. No changes to your game code.

Best for: Demos, showcases, single-player experiences, rapid prototyping.

Web Integration: Custom Portals

Use the YOM embed script (`yom-embed.js`) to build custom game portals with multiple streams, dynamic layouts, web-to-game communication, and branded onboarding flows.

Best for: Game portals, content aggregators, branded storefronts, interactive marketing pages.

UE5 SDK: Advanced Game Features

The YOM Unreal Engine 5 SDK adds responsive HUDs, bidirectional game-to-web events, control hints, multiplayer networking, voice chat, and Web3 features (inventory, airdrops, gated access, portals).

Best for: Studios building interactive multiplayer experiences with deep web integration.

Whitelabel Infrastructure: Bring Your Own Surface

YOM stays neutral at the application layer. Telco portals, smart-TV launchers, and publisher whitelabels all settle through the same network, so you can own the player relationship and the branding while YOM runs the streaming layer underneath. Build on the substrate, whoever owns the player.

Best for: Telcos, OTT operators, and publishers distributing a catalog under their own brand.

How the Build Paths Map to Walter's Motions

These developer paths plug into the three commercial motions that **Walter Limited** runs on the YOM network. The how-to above is the same; the motion is how you take it to market and how billing works.

- **Discovery** powers free embedded demos. Drop a playable demo on Reddit, Discord, Telegram, a landing page, or an ad unit using Quick Start or Web Integration. No service fee; sessions run at base compute cost. This is the wedge that turns a click into a player.
- **Kingmaker** is B2B Self-Service Hosting at **$89 per slot per month** (100 sessions included; overage billed per session at the tier rate plus a 20% markup). Sign up, upload a build, configure a storefront, go live. No enterprise sales cycle. Most studios use Quick Start or the UE5 SDK here.
- **Vampire** is the consumer **Compute Pass** (**$9.99/month**), where players link Steam or Epic and stream games they already own (bring-your-own-game). Publishers route their player base to the Compute Pass as the streaming layer, while keeping their own store and revenue.

Whichever motion fits, the integration work on this page is what gets you there: YOM is infrastructure, not a storefront taking a cut. You keep control of monetization, whether that is a subscription, a premium title, an ad-supported demo, or a free showcase.

Design for Instant Play

Cloud-streamed games live on webpages and are instantly accessible from any device. To make the most of this paradigm:

- **Skip main menus:** players expect to jump straight into gameplay.
- **Design responsive:** your game will run on phones, tablets, and desktops.
- **Think web-native:** use web widgets and bidirectional game-to-web communication instead of in-game UI where it makes sense.

quick start

Get a game streaming on YOM in two steps: upload your binary, then embed an iframe.

> **Network status:** YOM is in network launch, with the testnet live since June 2025; beta mainnet rolling out June 2026. The Quick Start flow below is live today on the available network. If a step looks different in the dashboard as the network scales, contact us at info@yom.net.

Step 1: Deploy Your Stream

1. Compile your game to **Linux x86_64**. Any engine works: Unreal Engine, Unity, Godot, or custom.
2. Log in to app.yom.net/streams.
3. Create a new stream:
   - **Stream Name**: a unique identifier (e.g., `my-game-demo`)
   - **Build Target**: choose your target environment (e.g., Development)
   - **Upload Binaries**: drag and drop your compiled game files
   - **Choose Network**: select your preferred network (e.g., Test Net)
4. Review your configuration and click **Pay + Deploy**.
5. Wait for both server and client files to finish uploading. Do not refresh or leave the page during upload.

Once deployed, you'll receive a **Game ID** and **Stream ID**.

Step 2: Embed the Stream

Add this iframe to your webpage:

```
<iframe
  src="https://player.yom.net?gameId=YOUR_GAME_ID&streamId=YOUR_STREAM_ID&origin=https://yoursite.com"
  width="1280"
  height="720"
  frameborder="0"
  allowfullscreen
  allow="autoplay *; microphone *; clipboard-write *; pointer-lock"
  style="border: none;">
</iframe>
```

Replace `YOUR_GAME_ID`, `YOUR_STREAM_ID`, and `https://yoursite.com` with your actual values.

That's it. Players visit your page, click play, and the game streams directly to their browser.

Optional iframe parameters

Append these to the iframe `src` URL as query parameters:

Parameter	Values	Description
`guestName`	string	Pre-fill the player name
`requestMic`	`true` / `false`	Request microphone access
`forceMic`	`true` / `false`	Require microphone to proceed
`fullscreen`	`none` / `window` / `native`	Fullscreen mode (`window` works on iOS)
`autoInitiate`	`true` / `false`	Skip the play button, start immediately
`requiresName`	`true` / `false`	Require player name input (default: true)
`lang`	ISO-639-1 code	Language code (e.g., `eng`)
`publisher`	string	Publisher name for analytics
`coppa`	`0` / `1`	COPPA compliance flag
`genre`	string	Game genre for ad targeting

Security: CSP headers

If your site uses Content Security Policy headers, allow the YOM player origin:

```
Content-Security-Policy: frame-src 'self' https://player.yom.net;
```

Or via a meta tag:

```
<meta http-equiv="Content-Security-Policy" content="frame-src 'self' https://player.yom.net;">
```

Troubleshooting

If you encounter issues:

1. Check your browser console for error messages.
2. Verify your `gameId` and `streamId` are correct.
3. Ensure your CSP headers allow `frame-src https://player.yom.net` .
4. Confirm your game binary was compiled for Linux x86_64.

What's Next?

- **Web Integration:** Add custom UI, multi-stream pages, and web-to-game communication using the YOM embed script.
- **UE5 SDK:** Add responsive HUDs, bidirectional events, multiplayer, and Web3 features (optional).

For assistance, contact us at info@yom.net.

js integration

The YOM embed script (`yom-embed.js`) gives you full control over the stream experience. It manages iframe creation, player lifecycle events, fullscreen behavior, visibility toggles, and bidirectional communication between your webpage and the game.

Getting Started

Include the embed script on your page:

```
<script src="https://cdn.yom.net/yom-embed.js"></script>
```

Then add the required HTML elements:

```
<!-- Player container: the iframe will be injected here -->
<div class="yom-player-placeholder"
     data-yom-stream-id="1"
     data-game-id="YOUR_GAME_ID"
     style="width: 100%; aspect-ratio: 16/9;">
</div>

<!-- Launch button -->
<input class="yom-name-input" data-yom-stream-id="1" placeholder="Your name">
<button class="yom-launch-button" data-yom-stream-id="1">Play</button>
```

When the player clicks the button, `yom-embed.js` creates an iframe inside the placeholder div, connects to YOM, and starts the game stream.

Core Concepts

Required CSS classes

Class	Purpose
`.yom-player-placeholder`	Container where the stream iframe is created
`.yom-launch-button`	Button that initiates the stream
`.yom-name-input`	Optional text input for player name

Required data attributes

Attribute	On	Description
`data-yom-stream-id`	All elements	Links elements to a specific stream instance
`data-game-id`	`.yom-player-placeholder`	Your game's unique identifier

Optional data attributes

Set these on `.yom-player-placeholder` to configure behavior:

Attribute	Values	Description
`data-request-mic`	`true` / `false`	Request microphone access
`data-force-mic`	`true` / `false`	Require microphone to proceed
`data-fullscreen`	`none` / `window` / `native`	Fullscreen mode (default: `none`)
`data-auto-initiate`	`true` / `false`	Auto-start without button click
`data-require-name`	`true` / `false`	Require name input (default: `true`)
`data-hook`	CSS selector	Apply `.yom-active` class to this element during stream
`data-callback`	Function name	Custom validation before launch
`data-lang`	ISO-639-1	Language code
`data-publisher`	string	Publisher name
`data-coppa`	`0` / `1`	COPPA compliance
`data-genre`	string	Game genre

Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Security-Policy"
        content="frame-src 'self' https://player.yom.net;">
  <style>
    .yom-toggle-visibility { transition: opacity 0.3s ease; }
  </style>
</head>
<body>
  <!-- Stream container -->
  <div class="yom-player-placeholder"
       data-yom-stream-id="demo"
       data-game-id="my-game"
       data-fullscreen="window"
       style="width: 100%; aspect-ratio: 16/9;">
  </div>

  <!-- Pre-stream UI (hidden when stream starts) -->
  <div class="yom-toggle-visibility" data-yom-stream-id="demo">
    <input class="yom-name-input" data-yom-stream-id="demo" placeholder="Enter your name">
    <button class="yom-launch-button" data-yom-stream-id="demo">Start Game</button>
  </div>

  <!-- In-stream controls (shown when stream starts) -->
  <div class="yom-toggle-visibility" data-yom-stream-id="demo" style="display: none;">
    <button onclick="window.postToPlayer('demo', 'SpawnBot', {amount: '1'})">Add Bot</button>
    <button onclick="window.postToPlayer('demo', 'ClearBot', {})">Clear Bots</button>
  </div>

  <script src="https://cdn.yom.net/yom-embed.js"></script>
</body>
</html>
```

What's Next?

- **Custom Authentication:** Add validation before stream launch
- **Event Handling:** Send commands to the game and listen for game events
- **Dynamic Layouts:** Toggle UI visibility based on stream state
- **Easy UI Hooks:** Fullscreen modes, auto-scaling, and container hooks
- **Multi-Stream Setup:** Run multiple game streams on one page

custom authentication

You can run custom validation before the stream launches, for example checking a password, verifying a token, or gating access behind a form submission.

Setup

Add `data-callback` to the launch button with the name of your validation function:

```
<div class="yom-player-placeholder"
     data-yom-stream-id="1"
     data-game-id="my-game"
     style="width: 100%; aspect-ratio: 16/9;">
</div>

<input type="text" class="yom-name-input" data-yom-stream-id="1" placeholder="Your name">
<input type="password" id="password" placeholder="Enter password">
<button class="yom-launch-button"
        data-yom-stream-id="1"
        data-callback="validatePassword">
  Start Game
</button>
```

Define the Callback

Your function receives two parameters: `onSuccess` and `onFailure`. Call one of them based on your validation result.

```
function validatePassword(onSuccess, onFailure) {
    const password = document.getElementById('password').value;

    if (password === 'correct-password') {
        onSuccess();   // Stream launches
    } else {
        onFailure();   // Launch is cancelled, button resets
    }
}
```

The callback is invoked when the player clicks the launch button, **before** the iframe is created. If the function throws an error, the launch is cancelled and the error is logged.

Use Cases

- **Password-gated access:** Restrict streams to users with a code
- **Form validation:** Require fields to be filled before launch
- **External auth:** Call your own API to verify the user, then call `onSuccess()` in the response handler
- **Age verification:** Prompt for age confirmation before allowing play

event handling

YOM supports bidirectional communication between your webpage and the game stream. Send commands to the game and listen for events coming back.

Sending Commands to the Game

Use `window.postToPlayer()` to send commands to a running stream:

```
window.postToPlayer(streamId, commandName, data);
```

Parameter	Type	Description
`streamId`	string	Must match the `data-yom-stream-id` of the target stream
`commandName`	string	The function name registered in your game (see Registering Events)
`data`	object	Key-value pairs passed to the game function

Example: Bot Controls

```
<button onclick="window.postToPlayer('1', 'SpawnBot', {amount: '1'})">Add Bot</button>
<button onclick="window.postToPlayer('1', 'RemoveBot', {amount: '1'})">Remove Bot</button>
<button onclick="window.postToPlayer('1', 'ClearBot', {})">Clear All Bots</button>
```

Example: Camera Settings

```
window.postToPlayer('1', 'SetCameraSensitivityX', { MouseSensitivityX: '0.5' });
window.postToPlayer('1', 'SetCameraSensitivityY', { MouseSensitivityY: '0.5' });
```

Example: Stream Control

```
window.postToPlayer('1', 'pause', {});
window.postToPlayer('1', 'resume', {});
```

Listening for Events from the Game

The YOM player sends events to your page via `postMessage`. Listen for them on the `window` object:

```
window.addEventListener('message', function(event) {
    // Only trust messages from the YOM player
    if (event.origin !== 'https://player.yom.net') return;

    const { event: eventName, streamId, data } = event.data;

    switch (eventName) {
        case 'playerReady':
            console.log(`Stream ${streamId} is ready`);
            break;
        case 'streamActive':
            console.log(`Stream ${streamId} is now playing`);
            break;
        case 'streamClosed':
            console.log(`Stream ${streamId} ended`);
            break;
        case 'error':
            console.error(`Stream ${streamId} error:`, data.message);
            break;
    }
});
```

Available Events

Event	When it fires
`playerReady`	Stream iframe is loaded and ready
`streamActive`	Game has started, player is in the experience
`streamClosed`	Player exited or stream ended
`error`	Something went wrong
`requestFullscreen`	Player requested fullscreen
`exitFullscreen`	Player exited fullscreen

Google Tag Manager Integration

All events from the player are automatically pushed to `window.dataLayer` for GTM/GA integration:

```
// Automatically pushed by yom-embed.js
window.dataLayer.push({
    'event': 'streamActive',
    'category': 'yom-event',
    'streamId': '1',
    'sessionId': 'abc123'
});
```

Configure your GTM container to capture `yom-event` category events for analytics tracking.

Games built with the UE5 SDK can also push custom events directly to the data layer.

dynamic layouts

Show and hide page elements based on whether a stream is active. This lets you build pre-stream landing UIs that disappear when the game starts, and in-stream controls that appear during gameplay.

Visibility Toggling

Add the `.yom-toggle-visibility` class and a `data-yom-stream-id` to any element you want to toggle:

```
<!-- Visible before stream, hidden during stream -->
<div class="yom-toggle-visibility" data-yom-stream-id="1">
    <h2>Welcome! Click below to start playing.</h2>
    <button class="yom-launch-button" data-yom-stream-id="1">Play Now</button>
</div>

<!-- Hidden before stream, visible during stream -->
<div class="yom-toggle-visibility" data-yom-stream-id="1" style="display: none;">
    <p>Game is running. Use the controls below:</p>
    <button onclick="window.postToPlayer('1', 'SpawnBot', {amount: '1'})">Add Bot</button>
</div>
```

How it works:

1. When the stream becomes active (`playerReady`), elements toggle: visible ones fade out, hidden ones fade in.
2. When the stream closes (`streamClosed`), elements toggle back to their original state.

CSS transition

The fade animation is 300ms. Add this CSS so the transition is smooth:

```
.yom-toggle-visibility {
    transition: opacity 0.3s ease;
}
```

Container Hooks

Use `data-hook` on the `.yom-player-placeholder` to apply an `.yom-active` class to any container element when the stream is running:

```
<div id="game-section">
    <div class="yom-player-placeholder"
         data-yom-stream-id="1"
         data-game-id="my-game"
         data-hook="#game-section"
         style="width: 100%; aspect-ratio: 16/9;">
    </div>
</div>
```

Now you can style the active state with CSS:

```css
#game-section.yom-active {
    background: #000;
    padding: 0;
    max-width: 100%;
}
```

The `.yom-active` class is added when the stream starts and removed when it ends, giving you full CSS control over layout changes.

easy ui hooks

The YOM embed script handles fullscreen management, aspect ratio scaling, and stream lifecycle automatically. Here's how to configure each behavior.

Fullscreen Modes

Set the fullscreen mode via `data-fullscreen` on your `.yom-player-placeholder`:

```
<div class="yom-player-placeholder"
     data-yom-stream-id="1"
     data-game-id="my-game"
     data-fullscreen="window">
</div>
```

Mode	Behavior	iOS Support
`none`	No fullscreen (default)	N/A
`window`	CSS-based fullscreen (`position: fixed` , fills viewport)	Yes
`native`	Fullscreen API (true fullscreen, hides browser UI)	No (falls back to `window`)

Window mode is recommended for cross-platform support. It works on iOS (where the native Fullscreen API is not available) and handles ancestor CSS transforms that would otherwise break `position: fixed` .

Fullscreen toggle cycle

When the player requests fullscreen repeatedly, the modes cycle:

- Desktop: `none` → `window` → `native` → `none`
- iOS: `none` → `window` → `none` (native is skipped)

Automatic Scaling

The stream automatically scales to fill its container while maintaining the correct aspect ratio. When fullscreen mode changes or the window is resized, the embed script notifies the player of the new viewport dimensions, and the game resolution adjusts accordingly.

No configuration needed. This works out of the box.

Stream Lifecycle

When the stream is active, the embed script manages several things automatically:

1. **Iframe creation:** Created inside `.yom-player-placeholder` on launch, with correct permissions (`autoplay` , `microphone` , `clipboard-write` , `pointer-lock`).
2. **Opacity fade-in:** The iframe fades in over 300ms once ready.
3. **Visibility toggles:** Elements with `.yom-toggle-visibility` are toggled on stream start/close.
4. **Container hooks:** The element referenced by `data-hook` receives the `.yom-active` class.

5. **Viewport notifications:** On resize or fullscreen change, the player is notified of new dimensions.

Auto-Initiate

Skip the launch button entirely and start the stream as soon as the page loads:

```
<div class="yom-player-placeholder"
     data-yom-stream-id="1"
     data-game-id="my-game"
     data-auto-initiate="true">
</div>
```

The player name can be pre-filled via the `guestName` iframe parameter or the `.yom-name-input` field.

Observability

The embed script includes built-in Grafana Faro integration for real-user monitoring. Console logs, errors, and traces are automatically captured, with no setup required on your end.

Errors are also reported to the YOM backend for debugging purposes.

multi-stream setup

YOM supports multiple independent game streams on a single page. Each stream has its own container, launch button, and communication channel.

Setup

Give each stream a unique `data-yom-stream-id`:

```
<!-- Stream 1 -->
<div class="yom-player-placeholder"
     data-yom-stream-id="1"
     data-game-id="racing-game"
     style="width: 100%; aspect-ratio: 16/9;">
</div>
<input class="yom-name-input" data-yom-stream-id="1" placeholder="Name for Racing">
<button class="yom-launch-button" data-yom-stream-id="1">Start Racing</button>

<!-- Stream 2 -->
<div class="yom-player-placeholder"
     data-yom-stream-id="2"
     data-game-id="fps-game"
     style="width: 100%; aspect-ratio: 16/9;">
</div>
<input class="yom-name-input" data-yom-stream-id="2" placeholder="Name for FPS">
<button class="yom-launch-button" data-yom-stream-id="2">Start FPS</button>
```

Each stream operates independently, launching, stopping, and communicating separately.

Sending Commands to Specific Streams

Use `window.postToPlayer()` with the target stream's ID:

```
// Send command to stream 1
window.postToPlayer('1', 'SpawnBot', { amount: '1' });

// Send command to stream 2
window.postToPlayer('2', 'SetCameraSensitivityX', { MouseSensitivityX: '0.3' });
```

Listening for Events from Specific Streams

Events from the player include a `streamId` so you can distinguish which stream they came from:

```
window.addEventListener('message', function(event) {
    if (event.origin !== 'https://player.yom.net') return;

    const { event: eventName, streamId } = event.data;

    if (eventName === 'streamActive') {
        console.log(`Stream ${streamId} started playing`);
    }
});
```

Per-Stream Visibility Toggles

Visibility toggles are scoped to their `data-yom-stream-id`:

```
<!-- Only toggles when stream 1 changes state -->
<div class="yom-toggle-visibility" data-yom-stream-id="1">
    Controls for Racing Game
</div>

<!-- Only toggles when stream 2 changes state -->
<div class="yom-toggle-visibility" data-yom-stream-id="2">
    Controls for FPS Game
</div>
```

Use Cases

- **Game portals:** Showcase multiple games on a single landing page
- **Comparison pages:** Let players try two experiences side by side
- **Content aggregators:** Curate streams from different publishers
- **Events:** Run multiple tournament streams on one dashboard

ue5

> **The YOM UE5 SDK is optional.** Any game compiled to Linux x86_64 can stream on YOM without any SDK. See the Quick Start. Use the SDK when you need responsive HUDs, bidirectional web-to-game communication, multiplayer networking, voice chat, or Web3 features (inventory, airdrops, gated access, portals).

The UE5 section provides a comprehensive guide on setting up and delivering your project to get the maximum out of YOM. The SDK contains solutions and templates specifically for quickly iterating cloud gaming streams. A Checklist Before Starting (!):

- If you are opting to use YOM within an existing Unreal Engine project, please check the Linux support for existing plugins. If your plugin does not officially support compile to Linux, in most cases it still compiles to Linux and if not on the first try, the fix is usually very simple.
- Verify you have the right Clang version for cross-platform compiling. Install the cross-compile toolchain that you can find on the bottom of the page.
- You need to install visual studio to build the headers for the SDK.
- For multiplayer support you need to download and use the source version of Unreal Engine.

Step 1: Use a C++ UE5 Project

You need to use a C++ version of an UE5 project. This can be achieved by setting up a new project and selecting C++ as the project defaults or by converting your existing blueprint project to C++. There is no disadvantage for picking C++ over Blueprints as you can still use Blueprints throughout your project.

If you are converting from an existing project follow the following steps to convert it to a C++ project:

- Click Tools-> select “new C++ class”
- Select none as the parent class and press next.
- Press create and it will convert your Blueprint project to a C++ project.

Step 2: Add the SDK to the project

To incorporate the necessary SDK into your project, follow these steps:

- Close your project if you opened it as part of the previous step.
- Navigate to your root project folder and create a new folder called “Plugins”. This folder will serve as the container for the required plugins.
- Obtain the YOM UE5 SDK for your Unreal Engine version from the link. If you don’t have access, ask your YOM contact person. This will trigger a download of a ZIP file. The ZIP file will contain three plugins.
- Extract the zip file and copy and paste the three plugins into the “Plugins” folder you need to create in your project directory. Ensure that the plugins are directly placed inside the “Plugins” folder and not within any additional subfolders.

Result

By following the steps outlined above, you should now have a folder structure resembling the following:

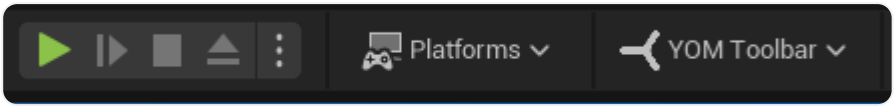
```
- YourProject
  - Plugins
    - YOM
    - VaRest
```

Step 3: Building the SDK

Now that everything is set, boot your project and compile the plugin headers. This need to be done once every time you update the SDK or add any other plugin and requires using a C++ project with Visual Studio installed. After it is set, it will automatically open the project.

Opening the project for the first time can take a bit of time. Just be patient. It will boot.

YOM SDK should be enabled by default. Validate by checking if a new toolbar appeared in your navigation bar called 'YOM Toolbar'.



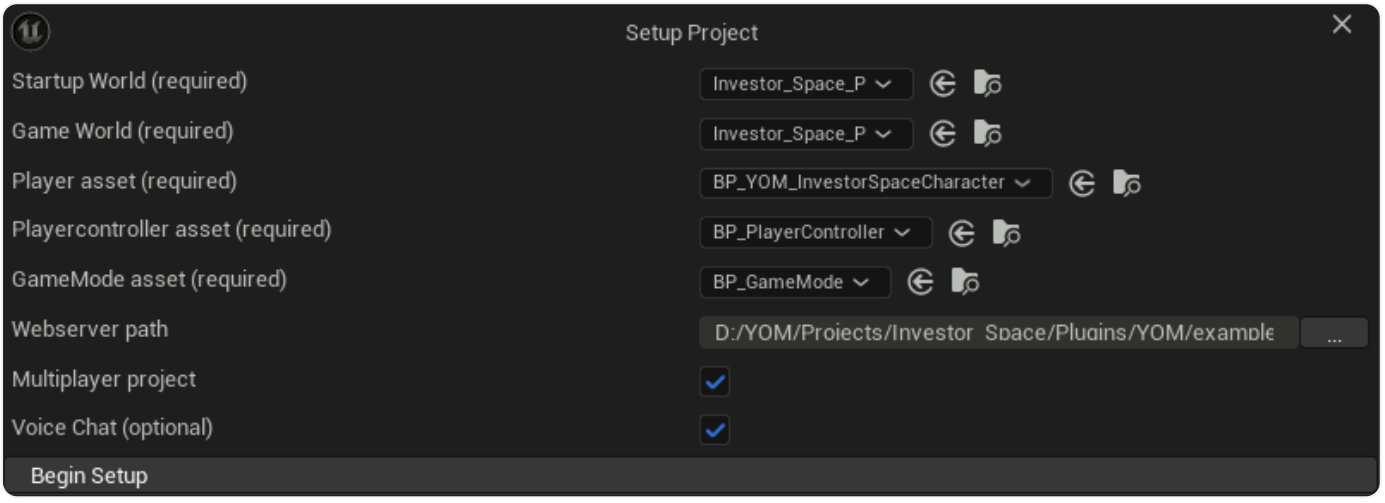
Follow these steps if the toolbar did not appear:

- Click on "Edit" and select "Plugins" from the dropdown menu. This will open the Plugins window.
- In the search bar within the Plugins window, type 'YOM' to search for the YOM Replicator SDK plugin.
- Enable it. If the plugin does not appear, try restarting the editor and repeat this step.
- Do the same for GltfRuntime and VaRest plugins in the same manner and enable them by clicking their respective checkboxes.
- After enabling the plugins, it is recommended to restart the editor. This allows Unreal to load the plugins properly.

Step 4: Setting up the SDK

To configure the YOM Replicator SDK for your project, follow these steps:

1. Open the setup screen by navigating to "Tools" in the top menu. Select "YOM" and then choose "Setup Project." This will open the setup window.



2. In the setup window, you will find several settings that need to be configured. Here is an explanation of each setting:

- **Startup World**: This setting sets the startup level. When we make a build this will be the starting point. As example it can be a mainmenu or if you don't have that the scene the game begins
- **Game World**: This setting sets the game level. This is the level where the game takes place.
- **Player Asset**: This setting defines the asset that players will control within the world. It can be any character or object that allows player interaction. The default character provided by the YOM plugin is the Yom_BP_ThirdPersonCharacter. Change it from "None" to "Yom_BP_ThirdPersonCharacter".
- **Playercontroller asset:** The Playercontroller that is used the game world or you want to use in the game world.
- **GameMode asset**: The GameMode that is used the game world or you want to use in the game world.
- **Webserver path**: The path to the webserver you want to use. We already provide an example webserver.
- **Multiplayer project**: If enabled, this setting allows the code from the SDK to be multiplayer. Enable this if you creating a game for multiplayer.
- **Voice Chat**: If enabled, this setting allows players to communicate with each other through voice chat within the experience.

Click "Begin Setup," and if the settings are correctly configured, a message will appear in the bottom right corner, instructing you to restart the editor. Click "Restart" to apply the changes.

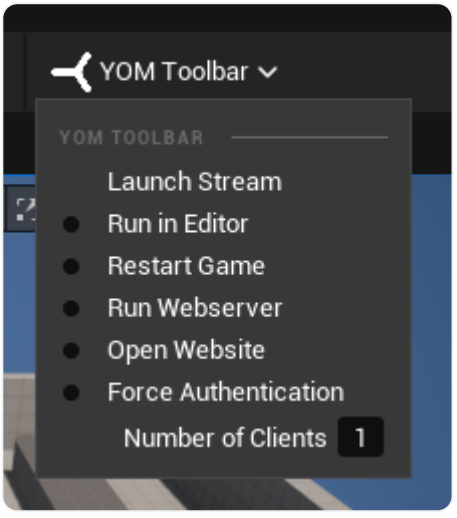
In case you want to make any further adjustments or verify the setup, you can run the setup tool again or manually modify the settings.

▶ Validate your configuration

Step 5: Debug Your Stream Locally

To test your game locally you can do this in three ways:

1. **Stream to localhost:8888.** This allows you to debug the full web experience, including player authentication. This is the default setting with all toggled turned off and is also the recommended setting to playtest the full experience.
2. **Play it in viewport.** If you want to quickly debug locally and you want to bypass the authentication window, you can opt for testing in the viewport. To enable this you need to toggle on 'Run in Editor' and 'Auto Spawn Player'.
3. **Play in standalone.** If you want to debug networking features without needing to first pass the authentication window in the web browser, you need to toggle on 'Auto Spawn Player'.



Toolbar

For these things we made the YOM Toolbar to easily test your game. We added some settings to help you test your game:

- **Launch Game.** Launches the game with the below settings.
- **Run in Editor.** When turned off it runs the game in stand alone and starts the signaling server. When turned on it will run the game in viewport and not start the signaling server.
- **Respawn Player.** When turned off it runs the game in stand alone and starts the signaling server. When turned on it will run the game in viewport and not start the signaling server.
- **Auto Connect Player:** When turned on it will execute the connect logic as you would connect from the pixelstreamer.
- **Run Webserver:** When turned on it will start the webserver when we press launch game.
- **Run Webclient:** When turned on it will start the webclient when we press launch game.
- **Force Authentication.** When true, it forces users to authenticate with their wallet else it will not work.
- **Number of Clients.** You can select the number of clients it needs to boot up. It will also boot up another signalling server. You can connect to it by adding a port. 8888 is the first one second 8889 etc.

The YOM Toolbar is to make it easy to test locally. You can still use the default buttons to play the game and the settings will apply to it.

Result

Pressing the Launch Stream button, the servers and games will boot and you can debug your game within a webpage. Make sure to add a Player Start to the level if the character does not spawn automatically.

Step 6: Package for YOM

- In Unreal Engine, go to the top bar: Tools -> YOM -> Package Project.
- In the opened menu, select the paths to your client and server build folders.
- If you used the recommended “Build” folder, paths should be automatically selected.
- Click “Package Project” to create separate zip files for the client and server.
- After the process completes, find a new folder called “BuildData” in your project directory.

Result

Inside “BuildData”, you’ll find two zip files, one for the client and one for the server, that are ready to upload to app.yom.net.

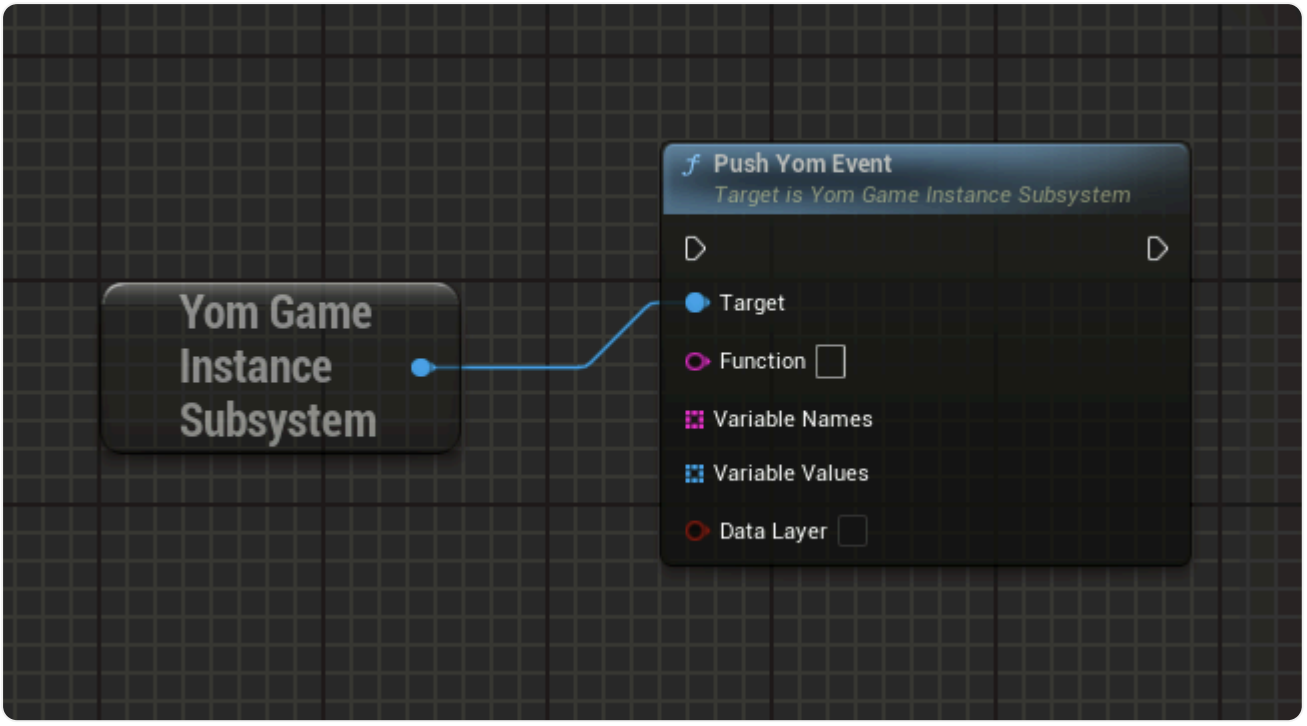
push events

The YOM SDK allows you to push your own custom events to the browser, allowing communication between the Unreal client and the pixel streamer / web browser. This guide explains how to set up, determine their event scope, and use them.

Step 1: Use PushYomEvent Function

Once you have the YOMGameInstanceSubsystem, use the PushYomEvent function:

```
YOMGameInstanceSubsystem->PushYomEvent(PEventScope, PFunction, PVariableNames, PVariableValues);
```



Use the “PushYomEvent” function with these parameters:

- Function: Name of the function to call
- Variable Names: Array of variable names
- Variable Values: Array of variable values
- DataLayer: Boolean to determine whether the event should be automatically pushed forwards to the DataLayer, used by marketing and data services Google Analytics.

Result

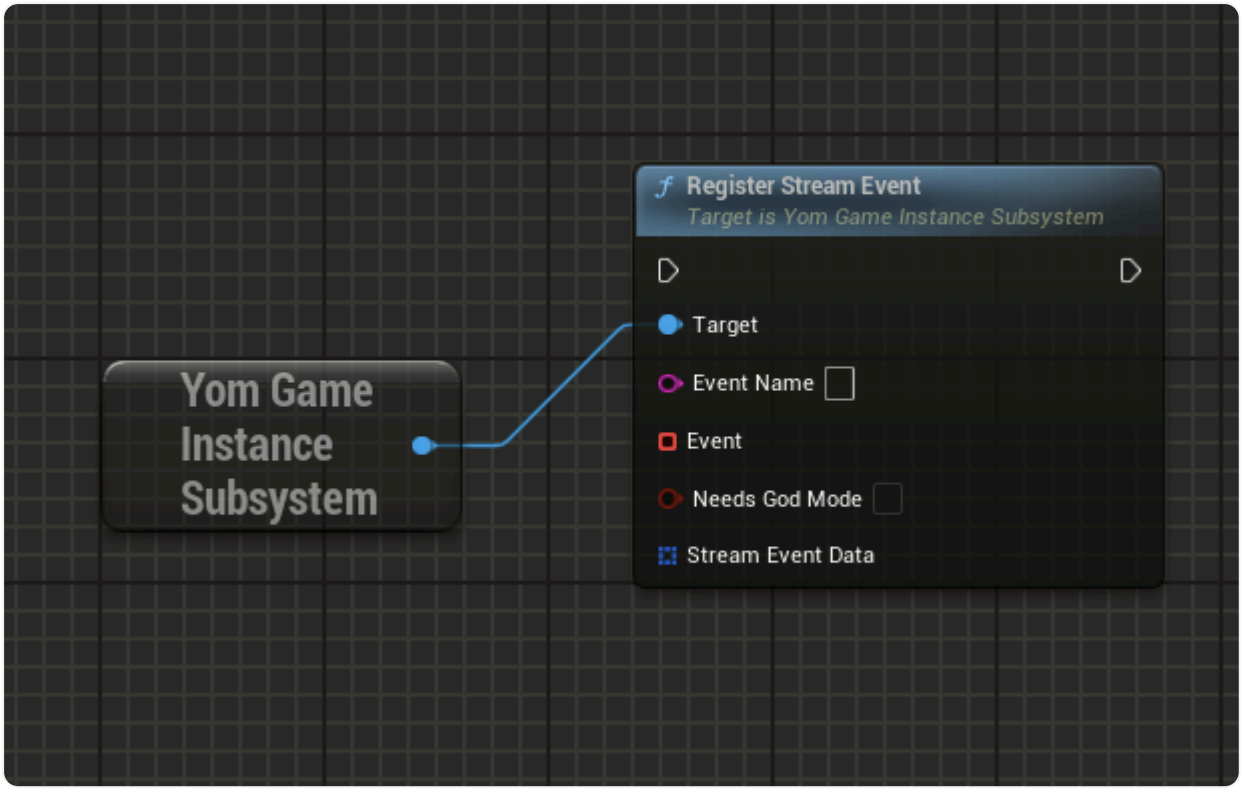
By following these steps, you can push communication from your Unreal Engine project to the pixelstreamer/web browser.

registering events

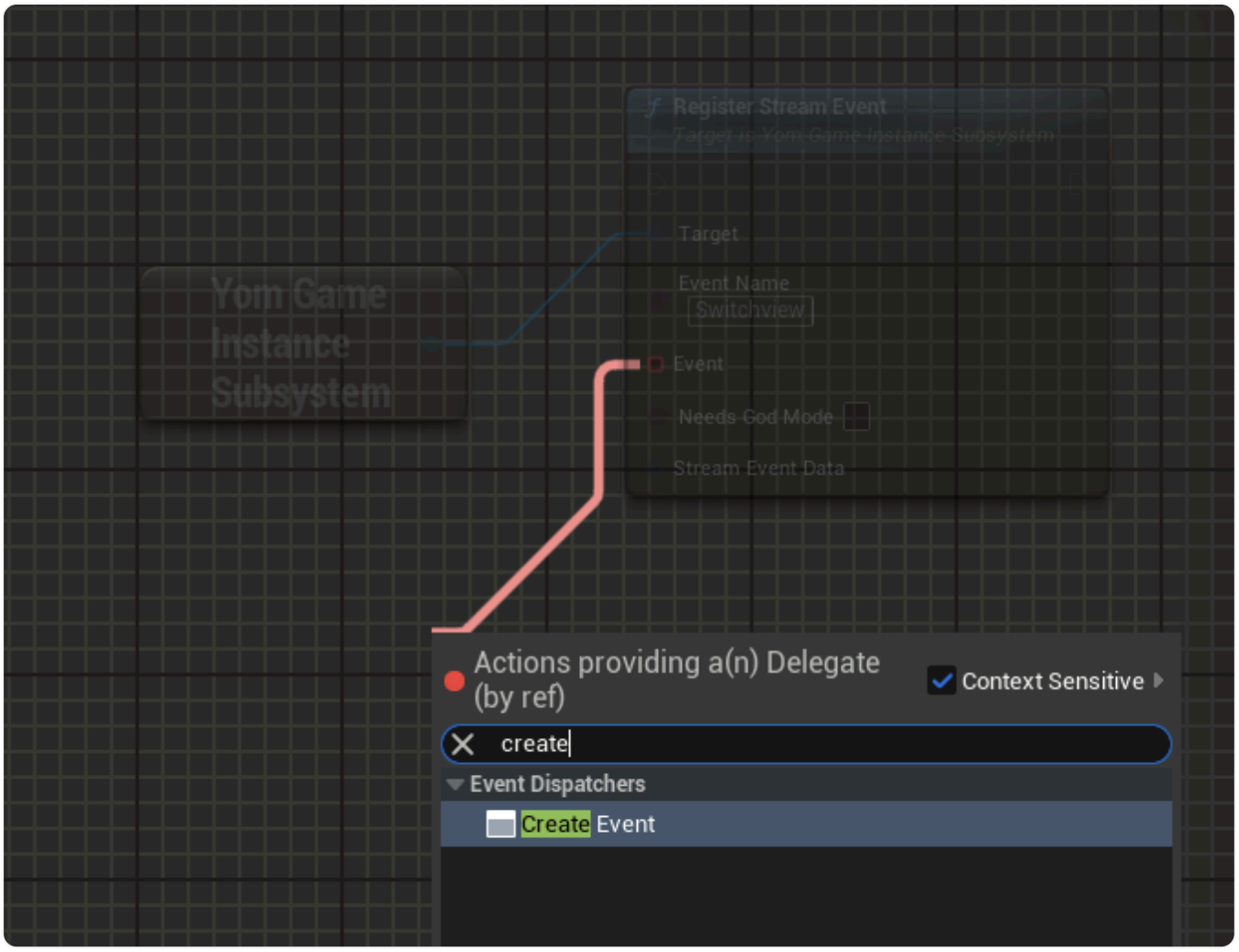
The YOM SDK allows you to create and listen for your own custom events, allowing communication between the Unreal client and the pixel streamer / web browser. This guide explains how to set up and use them.

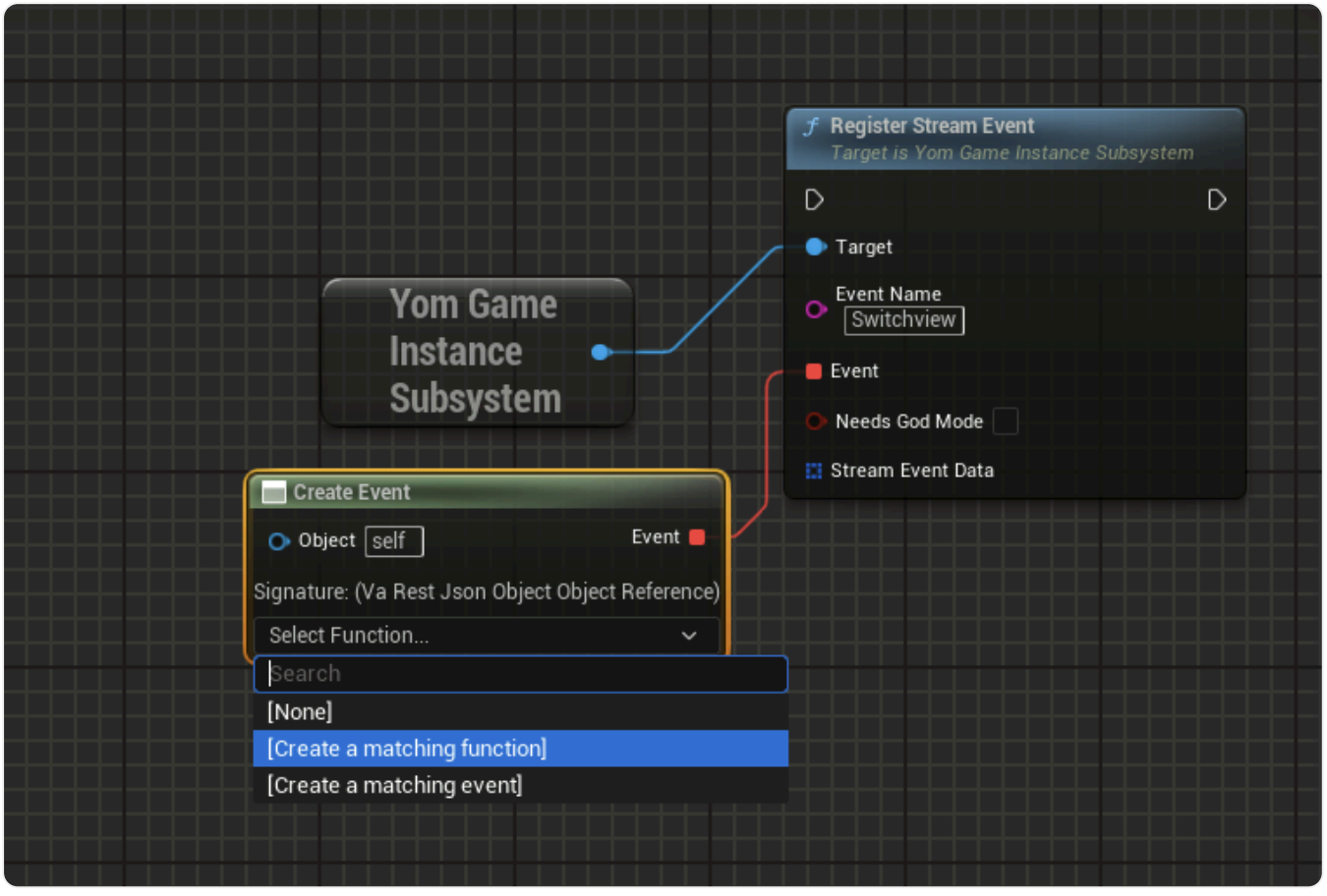
Step 1: Register a Stream Event

You get register a stream event by getting the YomGameInstanceSubsystem:

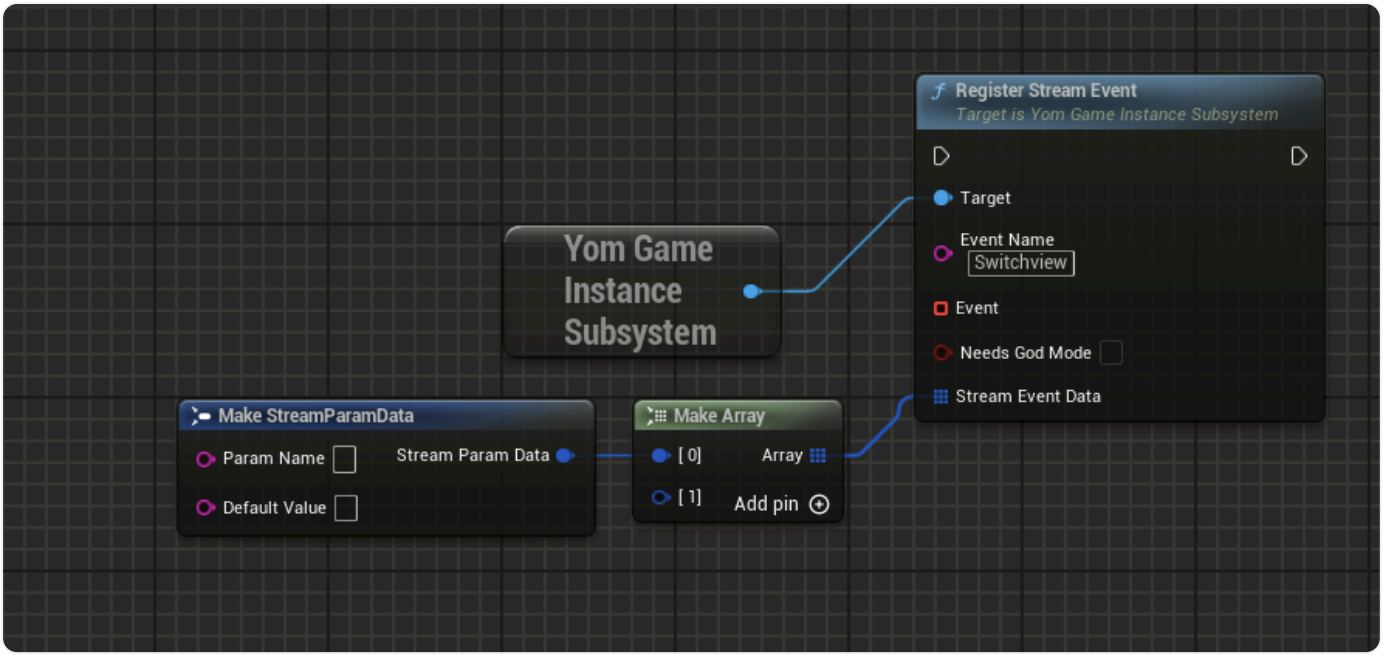


1. Set Event Name:
   - Name of the function to call
   - Will be displayed in the “GodMode” modal when enabled
   - Used to call the linked event
2. Set Event:
   - Event to be called when the function named FunctionName is triggered
3. Set NeedsGodMode:
   - Determines if the event is added to GodMode
   - Use for admin-only functions (GodMode requires a password)
4. Create the event:
   - Drag from Event and search for “create event”
   - Select “Create Event” node
   - Choose “select a function” and pick from the dropdown or create a new one


Register Stream Event
Target is Yom Game Instance Subsystem
Target
Event Name
Switchview
Event
Needs God Mode
Stream Event Data
Yom Game Instance Subsystem
Actions providing a(n) Delegate (by ref)
Context Sensitive
create
Event Dispatchers
Create Event


Register Stream Event
Target is Yom Game Instance Subsystem
Target
Event Name
Switchview
Event
Needs God Mode
Stream Event Data
Yom Game Instance Subsystem
Create Event
Object
self
Event
Signature: (Va Rest Json Object Object Reference)
Select Function...
Search
[None]
[Create a matching function]
[Create a matching event]

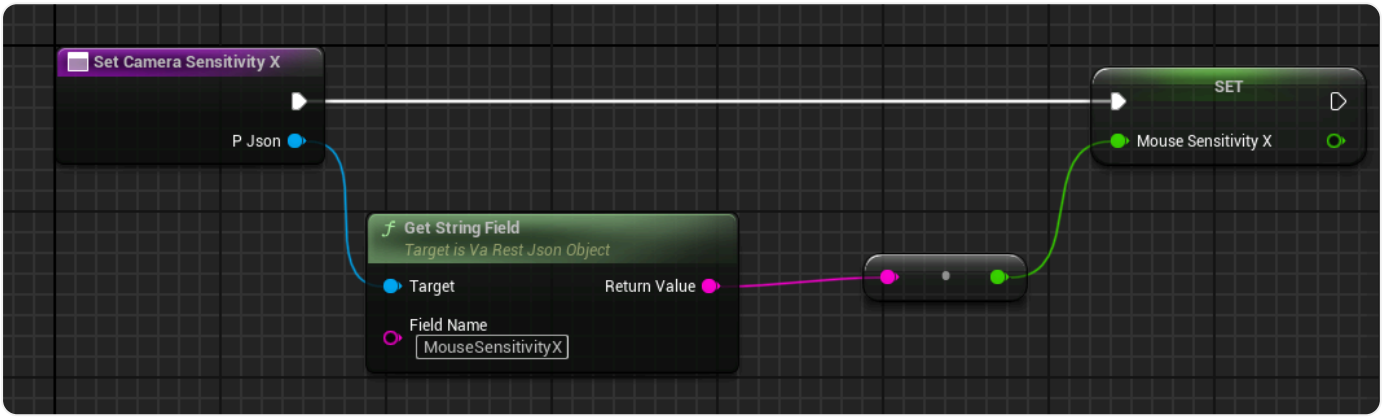
5. Set stream event data:
    - When you have variables that need to be passed in your function you can already define what names they have so it will show up in the godmode.
    - You need to create an array and add the amount of variables you have to that.
    - Param Name: name of the param variable.
    - Default Value: the value it needs to have or has at the start.



Note: The function input will be a Va Rest Json object, allowing data to be sent with the event.

Step 2: Handling Event Callback

To process data that is sent with the event:



1. Access the data object passed to the function
2. Use “get field by name and type” to retrieve specific variables
3. Implement your logic using the retrieved data
4. Note that everything we send will be a string and you need to convert that to the type you need.

responsive huds

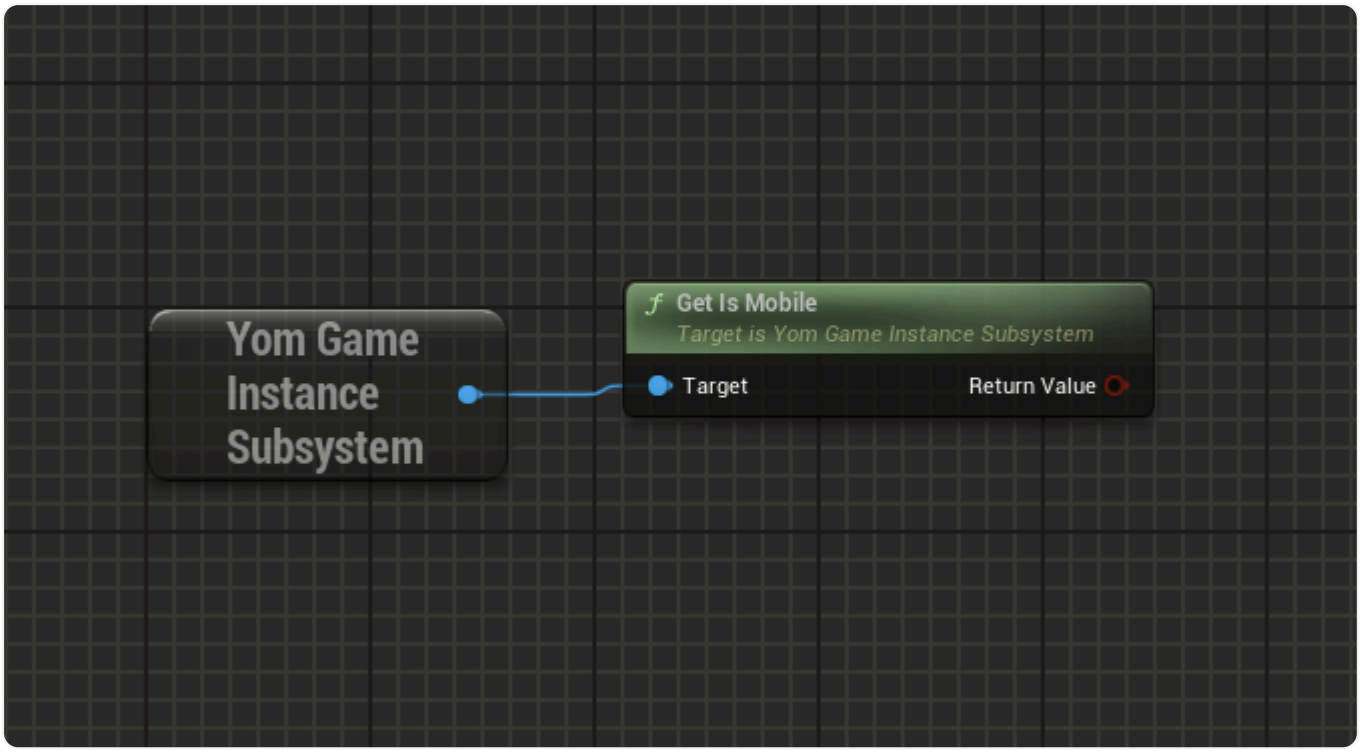
In the era of multi-device gaming, it's crucial to ensure your game is accessible and optimized for various platforms, including mobile devices and tablets. While YOM uses pixel streaming to deliver high-quality graphics across devices, understanding and adapting to different screen sizes and input methods can greatly enhance the user experience. Why implement responsive design:

1. Broaden your audience by supporting multiple device types
2. Ensure consistent gameplay experience across platforms
3. Optimize UI and controls for touch-based interfaces
4. Improve accessibility for players with different device preferences

Step 1: Detect Mobile Devices

YOM provides a built-in method to identify if a player is connecting from a mobile device:

1. Access the YomGameInstanceSubsystem in your Blueprint or C++ code.
2. Check the "GetIsMobile" boolean.



Step 2: Adjust for Mobile

Once you've detected a mobile device, you can make appropriate adjustments:

- Increase button sizes for touch input
- Implement different widget blueprints for mobile and desktop
- Utilize Size Boxes and Scale Boxes for flexible layouts
- Use anchors and alignment for position-independent layouts
- Simplify certain menus for smaller screens

- Implement touch-friendly controls

Step 3: Test with Chrome's Device Mode

To debug and test your mobile layout:

1. Open your game in Chrome
2. Press F12 to open Developer Tools
3. Click the "Toggle device toolbar" icon or press Ctrl+Shift+M
4. Select different device presets or set custom dimensions
5. Test your game's responsiveness in real-time

Result

By implementing these responsive design techniques, your YOM-powered game will provide a smooth, adaptable experience across a wide range of devices, from high-end desktop computers to smartphones and tablets.

Note: Always test your responsive design on actual devices when possible, as emulators may not perfectly replicate real-world conditions.

control hints

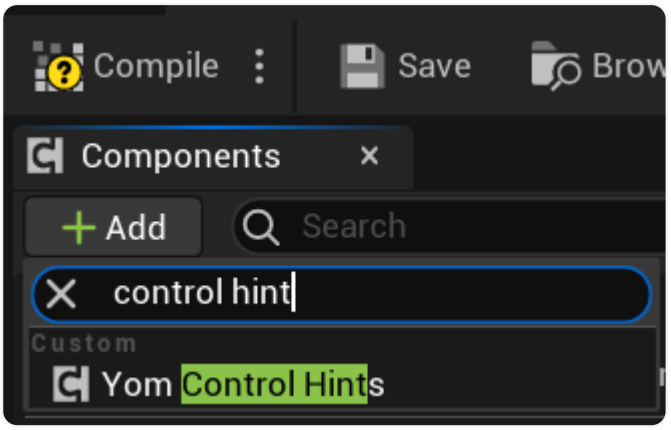
Displaying the Control Hints Panel to the player's screen

The YOM SDK allows creators to display a UI widget that would allow the player to understand the basic game controls. This guide will explain how to display and customize your own Control Hints with our ecosystem.

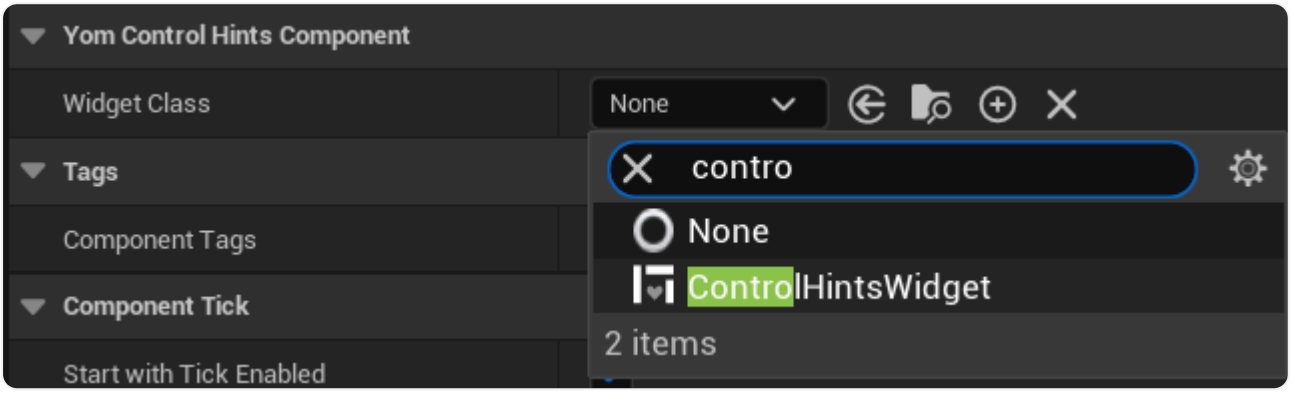
Step 1: Adding Control Hints component

To use the features of the Control Hints component, it needs to be connected to your player. This should be done by default. However, if you do not see YomControlHints under your player as a component, follow this step. Otherwise, you can skip to step 2.

YomControlHints is a component therefore we need to add it to the player blueprint. To add the component you will have to click add on component menu and search for `Yom Control Hints`.



After this, you need to assign a widget to the YomControlHints component. By default an existing widget should be already available, otherwise you would have to create one from scratch (step 2).



Example of ControlHints

characters

This guide explains how to create and customize a player character for your YOM stream. It covers using YOM's pre-built character or creating your own, understanding existing functionality, and how to modify or extend character features.

Step 1: Create Your Player Character

1. Create a New Blueprint:
   - Right-click in the Content Browser
   - Select "Blueprint Class"
   - Search for "Yom_BP_ThirdPersonCharacter" to use YOM's existing character framework
2. Assign the Player Character:
   - After creating your blueprint, assign it as the player character for your project

Step 2: Understand Yom_BP_ThirdPersonCharacter

Yom_BP_ThirdPersonCharacter includes:

1. Movement controls (WASD keys)
2. Interaction system (E key, requires BP_Interact interface)
3. Camera movement (third-person and first-person)
4. Camera switching (Q key)
5. Crouching (C key)
6. Jumping (Spacebar)
7. Walking/Running toggle (Left Shift)
8. Touch support for mobile devices
9. Customizable settings (walk/run speed, starting view)

Step 3: Modify Functionality

To customize Yom_BP_ThirdPersonCharacter:

1. Override the function you want to change
2. To add functionality:
   - Call the parent function
   - Add your custom logic
3. To remove functionality:
   - Override the function
   - Leave it empty
   - Do not call the parent function

*Note: Overriding **preserves** changes during SDK updates*

Step 4: Manage Controls

Default controls use Unreal's Enhanced Input System and are mapped to a controls customization modal that end-users can alter from within the stream:

1. To modify:

- Create your own "InputMappingContext" in the content folder
- Add required defaults
- Create new key mappings for your project

Caution: Direct changes to default controls may be overwritten by SDK updates

Result

Following these steps allows you to create, customize, and manage your player character while leveraging YOM's existing functionality and maintaining compatibility with future updates.

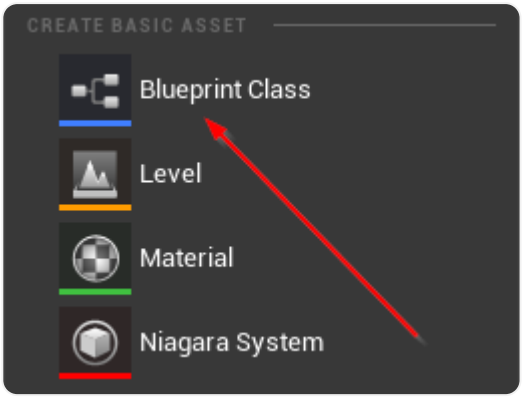
nameplates

Adding a nameplate to your character

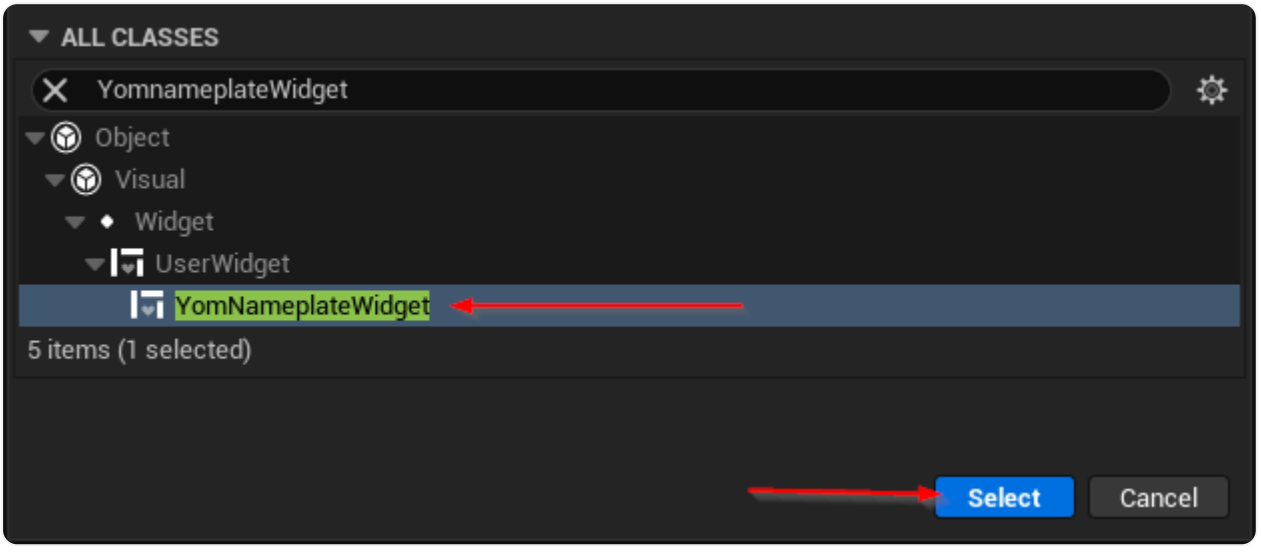
The YOM SDK allows creators to design their own nameplates. This guide will explain how to create and use your own nameplates with our ecosystem. If you just want to use our default nameplates, but it has not been enabled yet on your character you can move to Step 2 and use the `YomDefaultNameplateWidget` as your widget.

Step 1: Creating a custom nameplate

To create a custom nameplate you will have to create a new blueprint asset. Right-click in a folder in the `Content Browser` and under `Create Basic Asset` click on `Blueprint Class`.



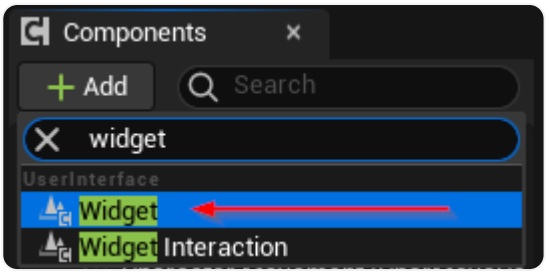
This will open up the blueprint dialog box in which you can search for `YomNameplateWidget`. Once you find the `YomNameplateWidget` select it.



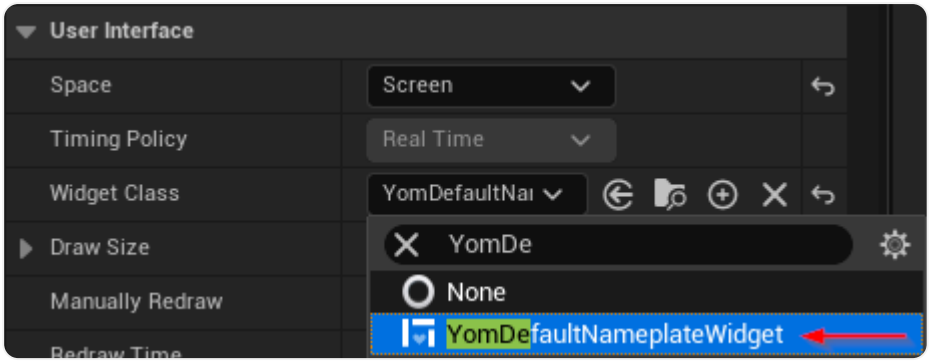
You can now customize your nameplate as you like, but one of the widgets must be a `TextBlock` widget with the name `WalletID`. In this `TextBlock` the walletID of the players will be printed when they are loaded into the game. Also give this walletID block a nice default name like `Unknown` for when the nameplate could not be loaded.

Step 2: Adding the nameplate to your character

Open up your character (this is not your `MetaspacePlayer`, but the character that the players will use to move around with) and under the `Components` tab add a `Widget` component.



Give this widget a nice name like `Nameplate` and select it. Under the `Details -> User Interface -> Widget Class` you can set your widget.

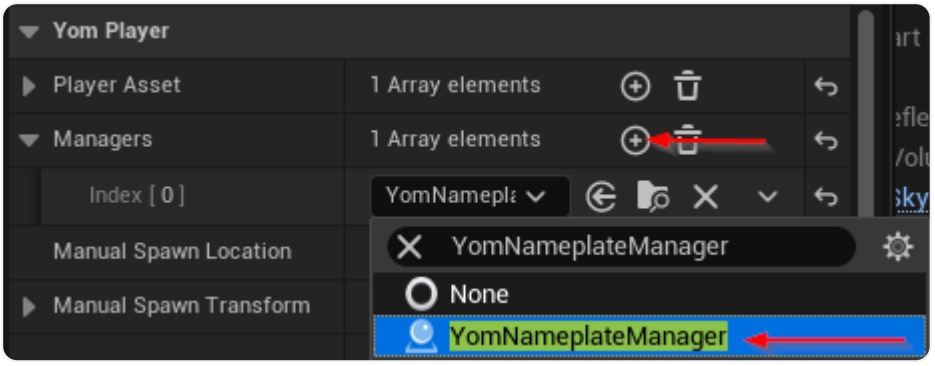


Tip: If you want the widget to always face the player, change `Space` from `World` to `Screen`.

In the `Viewport` tab, set the nameplate to the desired position for your character.

Step 3: Add the YomNameplateManager to your MetaspacePlayer

The nameplates will only work when you tell your `MetaspacePlayer` to load the `YomNameplateManager` in your game on startup. Go to your `MetaspacePlayer` and under `Details -> Yom Player -> Managers` add an element and insert `YomNameplateManager`.



You should now be able to see a nameplate when you spawn into the game.

counters

Adding a counter to your metaverse

The YOM SDK allows creators to design their own counters. This guide will explain how to create and use your own counters with our ecosystem.

Step 1: Adding counter actor to the world

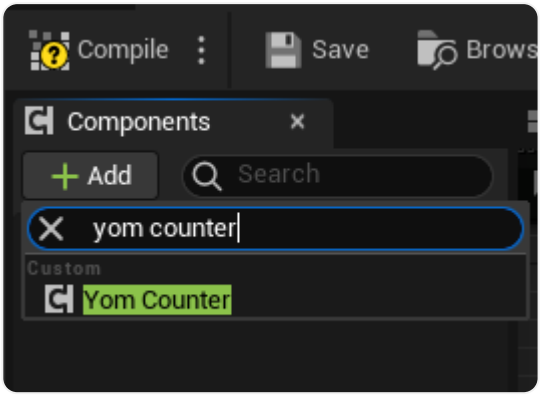
To use the features of the counter, it needs to be added to the world first.

YOM Counter is an actor and needs to be manually added to the world to start working. To add the actor, you will have to slide the actor from `Content Browser` under `Plugins -> YOM Content -> Counter` into your world.

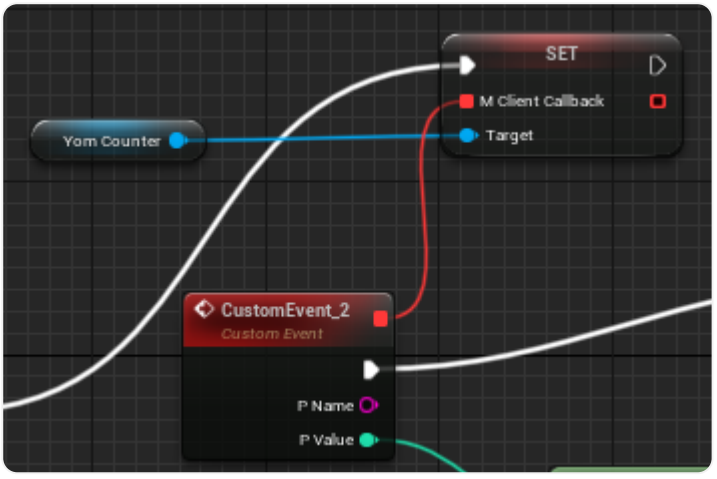
After this, you can call counter functions through blueprints without errors.

Step 2: Adding counter component to the player

To use the auto replicating feature of the counter (updating a counter value is shared with other clients), you will need to add the counter component to the player blueprint. You can do this by going to your `MetaspacePlayer` and using the add button on the top left corner of the screen.



After you do this, you can define what to do when a counter value is updated. This is done by using a callback event. This callback event gives you the name of the counter which is updated and its new value.

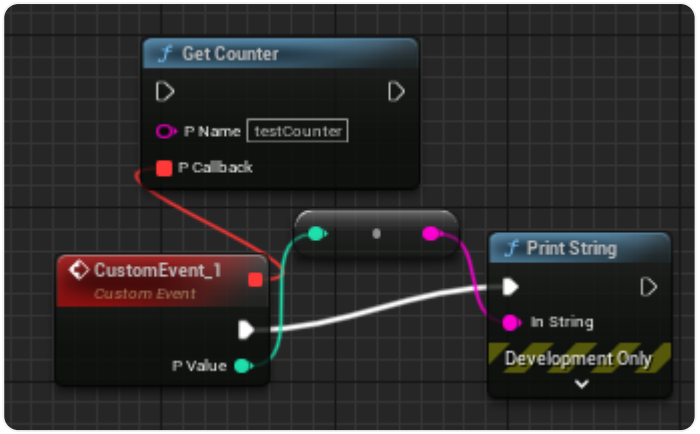


When a counter value is updated with `GetCounterPlayer` blueprint function, this callback function will be called on all of the clients with a counter component. Returned value and the name can be used however you like.

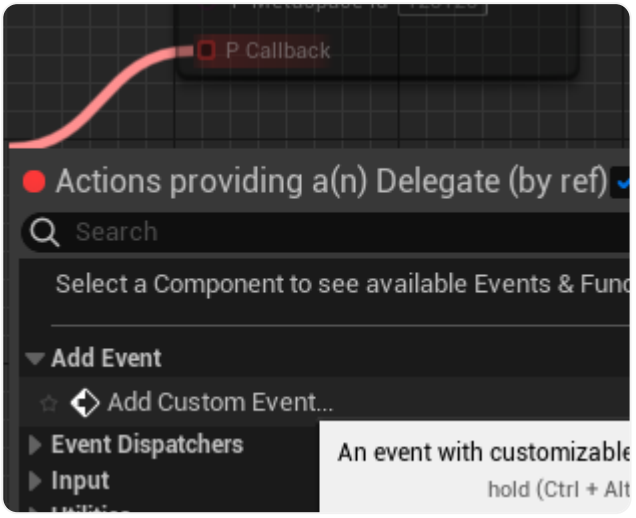
Step 3: Using blueprint functions to use the component

From the blueprint actions box (it is automatically open or you should right click to the screen), search for `Yom Counter` in order to see the available functions.

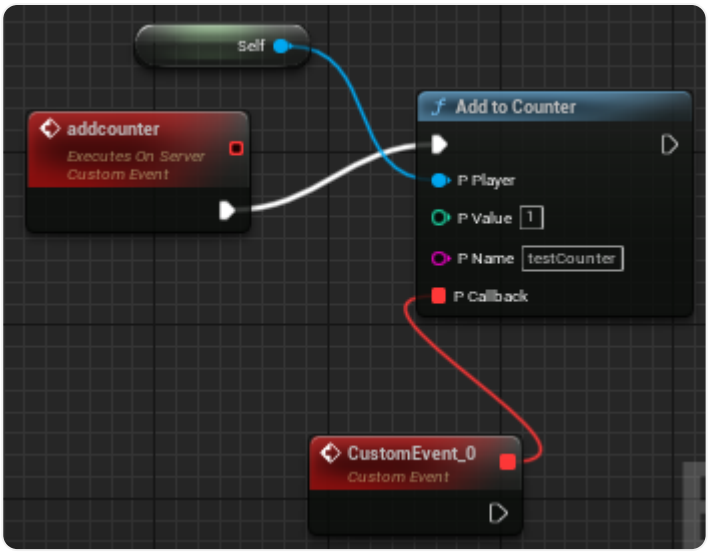
Example use case for `Get Counter` function:



In this example, we are trying to get `testCounter` in the current metaspace. Counter value is returned by a callback therefore we drag and drop P Callback node and create a custom event. P value returned by the event is the value of the counter which can be used however the user wants. In this example case it is printed to development screen.



Example use case for `Add to Counter` function:

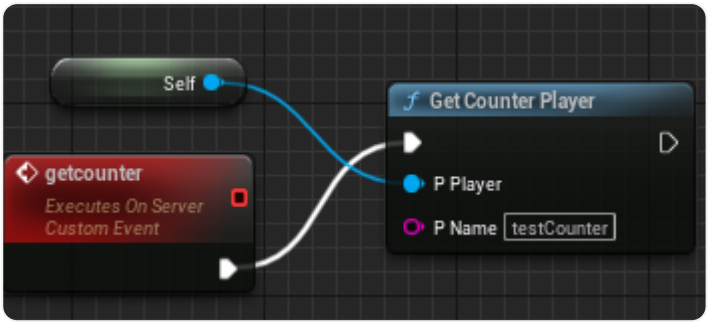


In this example, counter is set to increment the `testCounter` in the current metaspace by value 1. Therefore, this will increment that counter value by 1. P Callback function is the function which is returned after API Request is done. It can be used to execute more commands when your increment/decrement request is done.

If callback function is not required, a dummy callback function can be created as it can be seen from the image.

Finally, we can use `Get Counter Player` blueprint function in order to replicate the get request of the counter value. When it is called from a client, it makes a server call which is replicated to other clients as well. This way, all of the connected clients are getting the new value of the counter as well as the name of the counter.

Example use case for `Get Counter Player` function:



Note: we recommend not updating the counter too much, but ‘batch’ the counts, as every time the counter is updated it needs to be updated on the blockchain, which costs money.

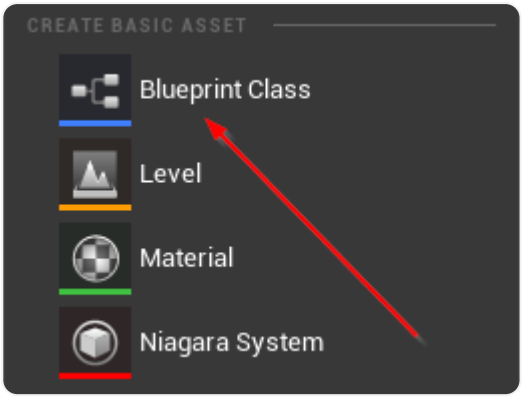
inventory

Setting up an inventory

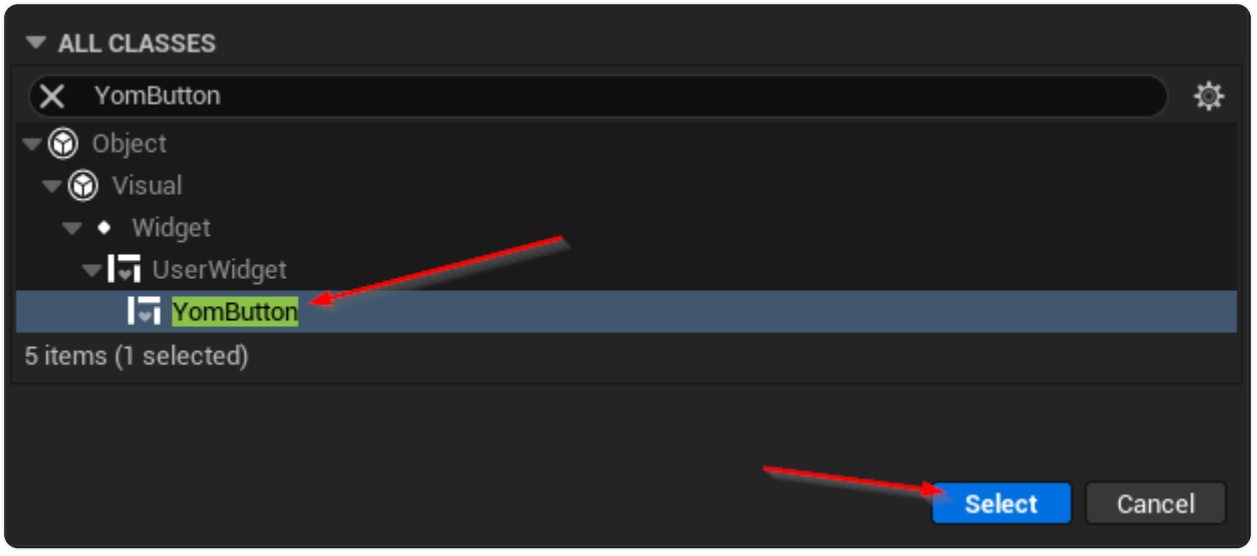
The YOM plugin allows creators to completely design their own inventory system. This guide will explain how to create your own inventory system and how to use your own inventory system in your metaspace.

Step 1: Creating your own buttons

The inventory system consists of buttons that can be pressed to spawn items. A single button needs to be designed for the system to work. To create a button you will have to create a new blueprint asset. Right-click in a folder in the `Content Browser` and under `Create Basic Asset` click on `Blueprint Class`.



This will open up the blueprint dialog box in which you can search for `YomButton`. Once you find the `YomButton` select it.



Name the button something like `MyButton` and open up the blueprint by double-clicking the new asset. In the blueprint editor, you will now have an empty screen, in which you will have to design your button. But before you start the button will need some widgets:

- `Canvas` (`Overlay`)
  - The canvas in which the button can be built
- `Button` (`Button`)

- The button that can be clicked

- `RarityWidget` (Any Widget)
  - The rarity widget, can be seen as the border around the button
- `LoadingWidget` (Any Widget)
  - The widget that is shown when the item is being loaded

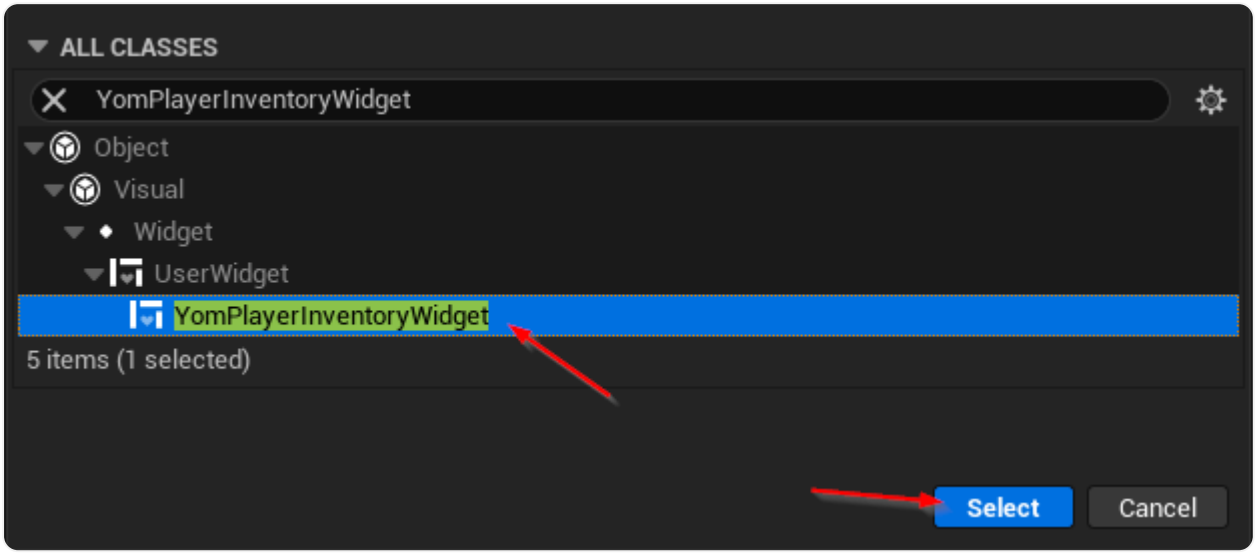
All of these widgets also need to have the name of the items described above. In the default `YomButton` the hierarchy looks like this:

Tips

- Set the visibility of the RarityWidget to `Not Hit-Testable (Self & All Children)` by clicking on the RarityWidget in the hierarchy and going to `Details -> Behavior -> Visibility`. This prevents the Rarity widget from 'overlapping' the button which would prevent a user from pressing the button.
- Design your button with a fixed size this will prevent the buttons from 'stretching' when loaded

Step 2: Creating your own Inventory Widget

Create another blueprint like in step 1, but this should have `YomPlayerInventoryWidget` selected.



Give this a name as well like `MyPlayerInventoryWidget` and open up the blueprint by double clicking the new asset. This will again give you an empty screen. On this screen, you can design your layout of the inventory. It will need one widget:

- BaseButtonWidget (A widget with slots for other widgets like the `Scroll Box`)
  - In this widget, your buttons will be created

The hierarchy of the default `InventoryWidget` looks like this:

Furthermore, your `MyPlayerInventoryWidget` needs to have some settings applied to it before it works. Click on `MyPlayerInventoryWidget` in the hierarchy and go to `Details -> Yom Inventory`. Here you will see the following settings:

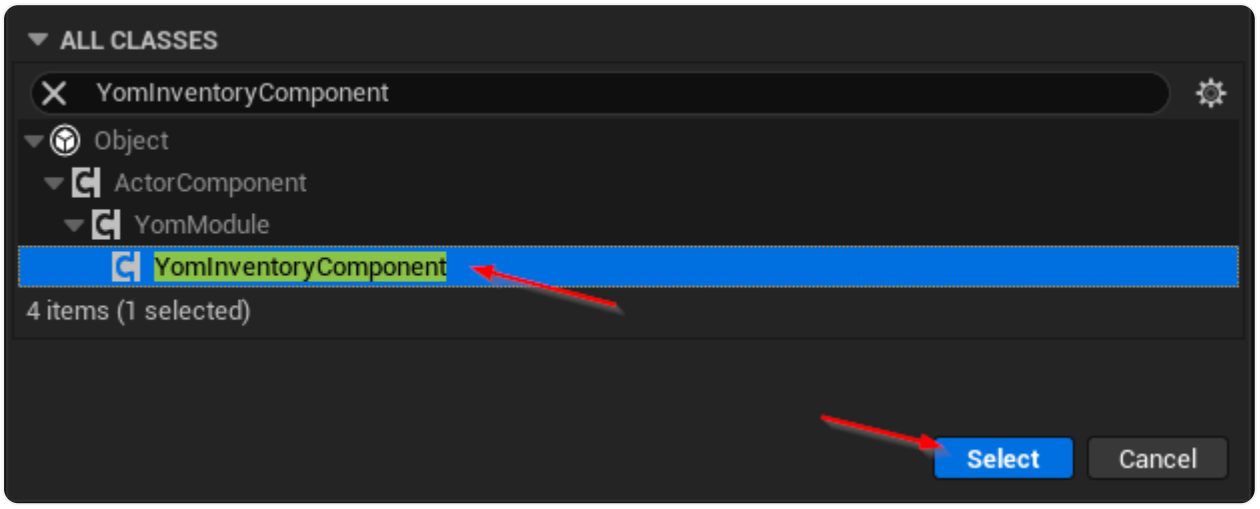
- `Yom Button Widget`
  - The widget for your button, set this to the button you created in step 1.
- `Base Button Widget`
  - This one should be set to the `BaseButtonWidget` in the hierarchy, but if it is not set, you can set it here
- `Start Button Amount`
  - This value tells the inventory system how many buttons it should spawn when the player is connected. If the player has more items than this amount more buttons will be loaded.

Tips

- Keep in mind that players could have more items in their inventory than you expect. Your inventory should not deform when a lot of items are being loaded.

Step 3: Creating a component

Create another blueprint like in step 1, but this should have `YomInventoryComponent` selected.



Give this a name as well like `MyInventory` and open up the blueprint by double clicking the new asset. In the asset, you will have to set one setting. Go to `Details -> Yom Inventory Component -> Inventory Widget` and set it to the widget you created in Step 2.

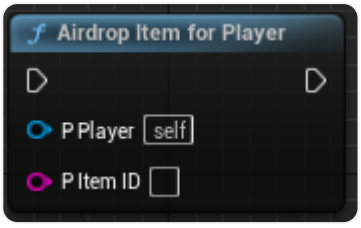
Step 4: Setting your inventory in your player

Open your `MetaspacePlayer` and under components remove `YomDefaultInventory` if present. Then add the component that you created in step 3. When you start the game now you can see your own inventory in action when pressing ‘I’ while playing.

airdrops

It is possible to incentivize and reward users in experiences by dropping items directly to their wallet as part of the experience mechanics. This should be done on the server side, and the player must have a connected wallet for it to work. Make sure the inventory is set up as well. In the airdrop node, you need to set two values:

1. YomPlayer: Specify the player to whom the item will be airdropped.
2. Item ID: Provide the ID of the item that will be airdropped. There are separate IDs for devnet and mainnet.

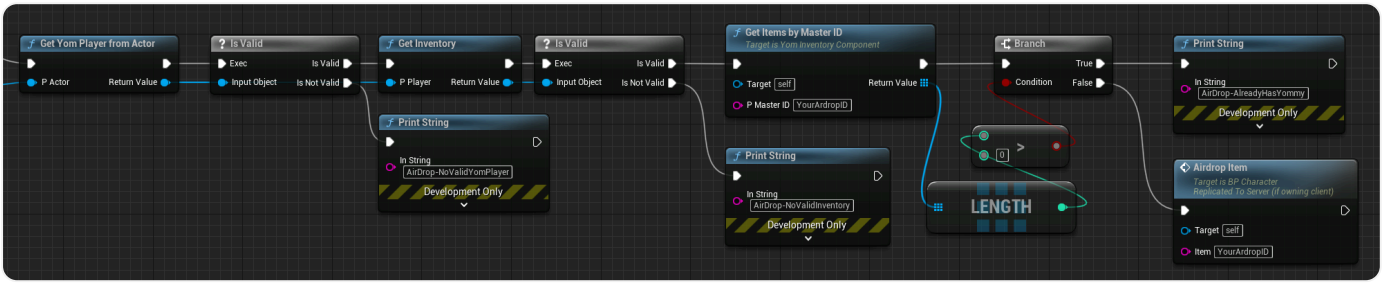


The airdrop node will add an item based on the item ID to the player's wallet. Please note that the airdrop function will not work for users who haven't logged in with a wallet.

Checking if airdrop received

If you want to check if a user has already received an airdrop, you can follow these steps:

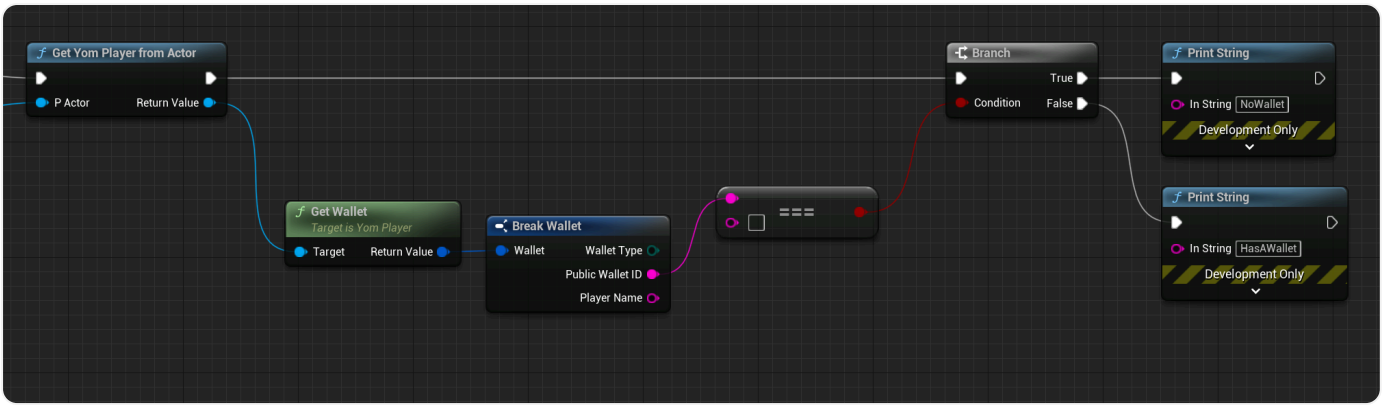
1. Retrieve the YomPlayer of the player.
2. Get the inventory of the YomPlayer.
3. Check if there is an item in the inventory with the ID of your master NFT ID. If an item with that ID exists, it means the user has already received the airdrop.



Checking wallet connection

To check if a user has a wallet connected, follow these steps:

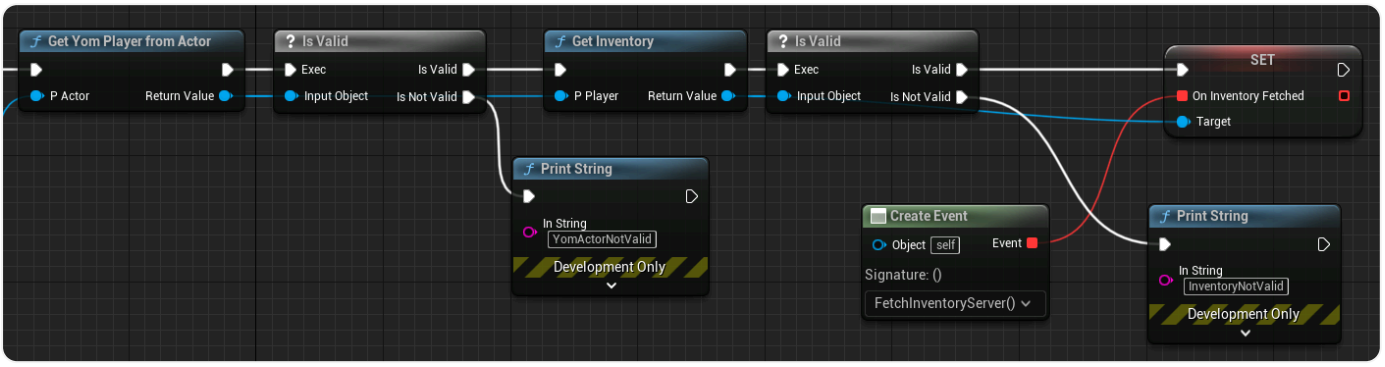
1. Get the player you want to check.
2. Retrieve the YomPlayer from that player.
3. Get the wallet of the YomPlayer.
4. Check if the wallet ID is empty. If it's not empty, it means the user has a wallet connected.



Checking if wallet is fetched

Fetching the wallet may take some time, so there is a callback function called “OnInventoryFetched” that you can use. To check if the wallet has been fetched, do the following:

1. Get the YomPlayer from the player.
2. Check if the YomPlayer has a valid inventory.
3. Bind an event to the “OnInventoryFetched” callback to be notified when the inventory is fetched.



Note: Airdrops last until the maximum supply of the item has been reached.

gated access

To create exclusivity and restrict access to experiences based on specific item ownership in YOM, you can follow these steps:

Step 1: Adding the Gated Access component

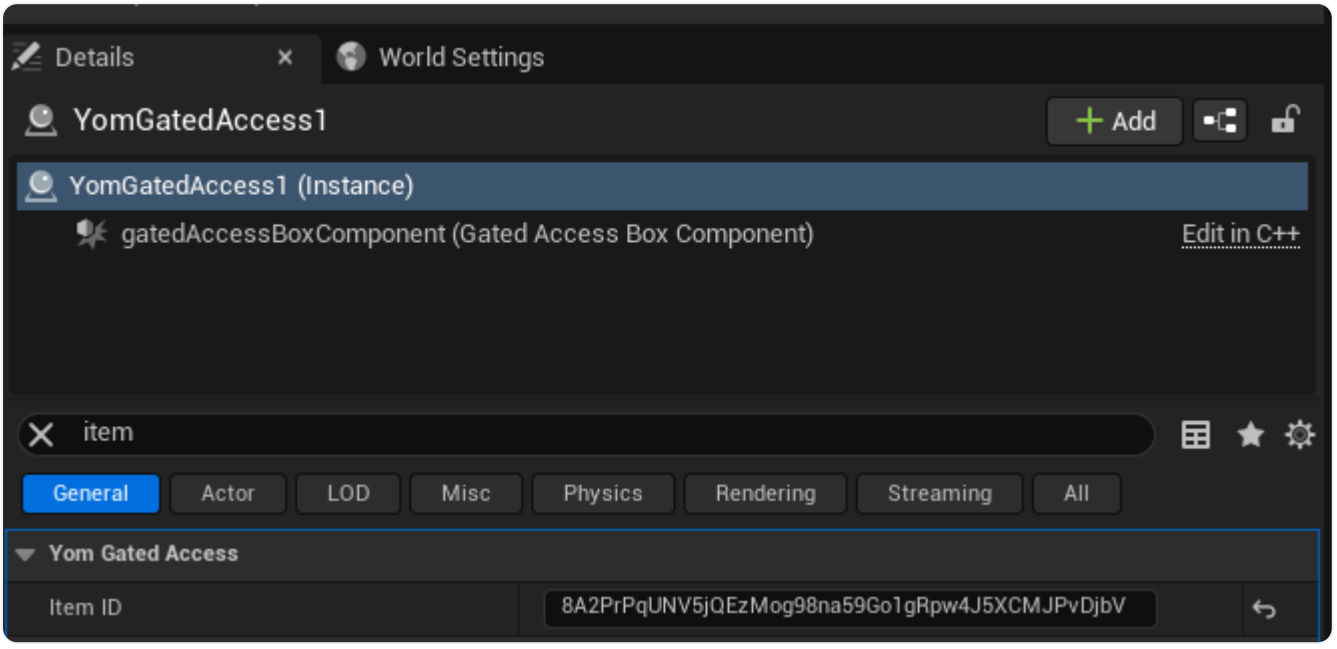
- By default, the Gated Access component (YomGatedAccess) should already be connected to your Metaspace Player. If you don't see it as a component, you'll need to add it manually.
- To add the component, click on the "Add Component" button in the blueprint editor and search for "Yom Gated Access." Add it to your Metaspace Player blueprint.

Step 2: Adding the Gated Access actor to the level

- To implement the Gated Access functionality, search for "Yom Gated Access" in the Place Actors window and drag it into the level where you want to create the restricted zone.

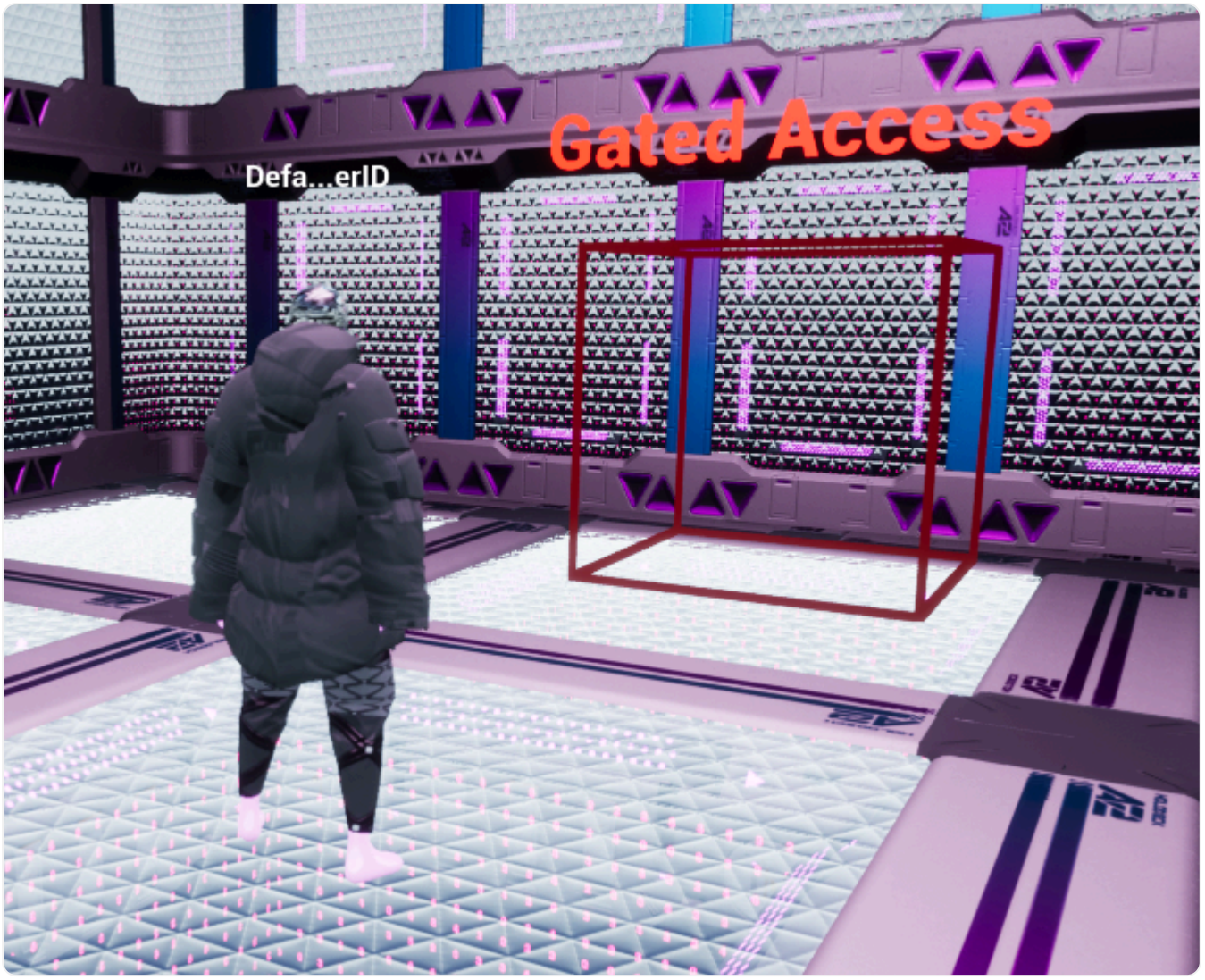
Step 3: Setting the Gated Access Item ID

- Click on the Gated Access actor in the level.
- Look for the "Item ID" property and insert the NFT address or master ID of the NFT item that is required for access. Note that you have separate IDs for devnet and mainnet.



Step 4: Setting up a Collision Layer

- Set up a collision object channel for the Gated Access to work properly.
- Go to Project Settings and navigate to the Collision section.
- Click on the "New Object Channel" button.
- Name the channel as "GatedAccess" and set the default response to "Overlap."



Notes:

- You can check if a player has access to the Gated Access zone by calling the blueprint function “Can Player Access.”
- Make sure the inventory is enabled for the player for this functionality to work.
- The user’s wallet needs to be fetched before they can go through the Gated Access.

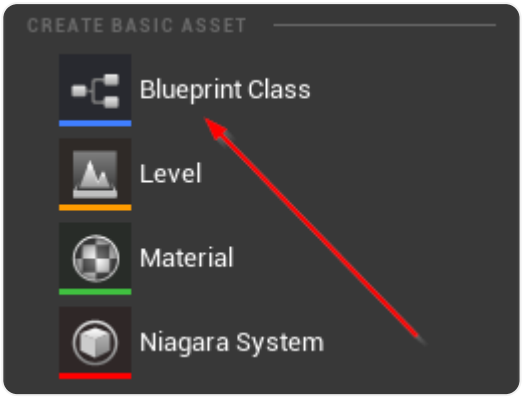
portals

Adding a portal to your Stream

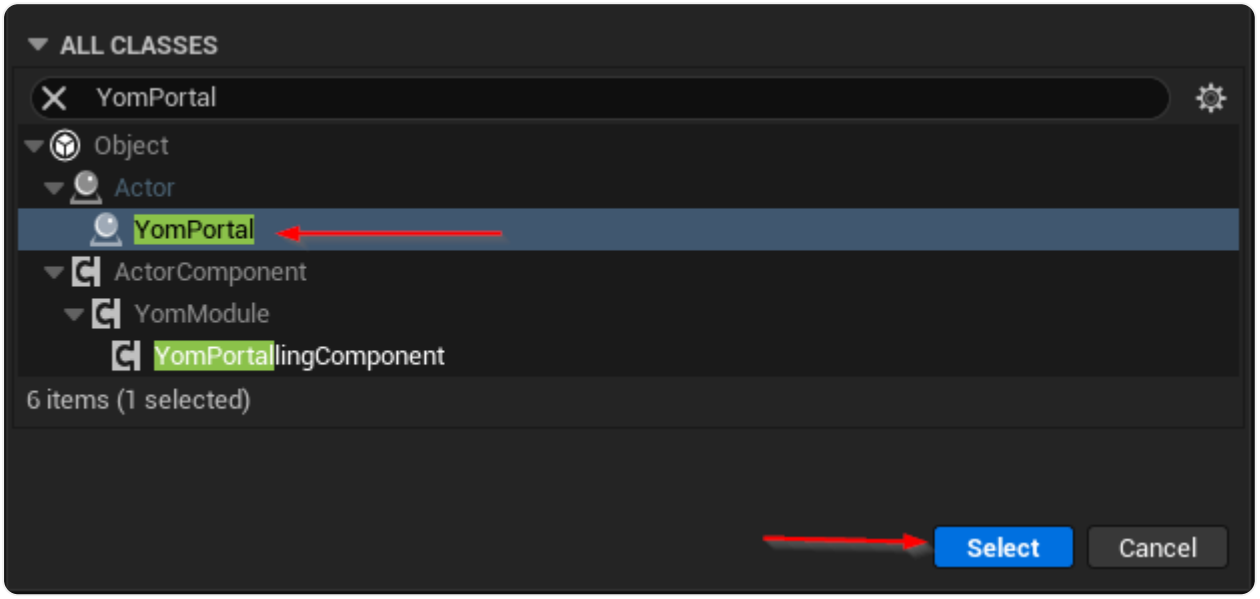
The YOM SDK allows creators to design their own portals. This guide will explain how to create and use your own portals with our ecosystem. If you just want to use our default portals, you can use the portals in the `Content Browser` under `Plugins -> YOM Content -> Portals` and skip to step 2. The portals that you can use are the `YomDefaultPortal` and the `YomFXPortal`, where the `YomFXPortal` is more visually appealing. Note that you must have the option `Allow teleporting` set to true, you can verify this by going into your `MetaspacePlayer` and checking if it has a `YomPortallingComponent`.

Step 1: Creating a Custom portal

To create a custom portal you will have to create a new blueprint asset. Right-click in a folder in the `Content Browser` and under `Create Basic Asset` click on `Blueprint Class`.



This will open up the blueprint dialog box in which you can search for `YomPortal`. Once you find the `YomPortal` class select it.



You can now start to customize the portal in every way you like. If you want the plugin to automatically set the material you must set `Details -> Auto Set Portal Material` to true and the portal must have a `StaticMeshComponent` called `PortalDisplay`, this will automatically edit the material of the `StaticMeshComponent` at runtime when you

connect to a different stream. If you want to set the material yourself you can use the `Get Portal Material` node in the `Event On Portal Open` and `Event On Portal Close` events. Furthermore the portal will need a `Collision` component for triggering the portal behaviour and you can set a default material under `Details -> Materials` for when the portal is not connected.

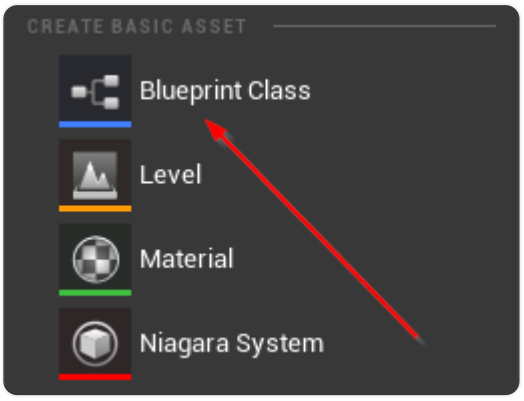
Step 2: Adding the portal to your level

You can now add the newly created portal to your level by taking it from the `Content Browser` and sliding it into your level.

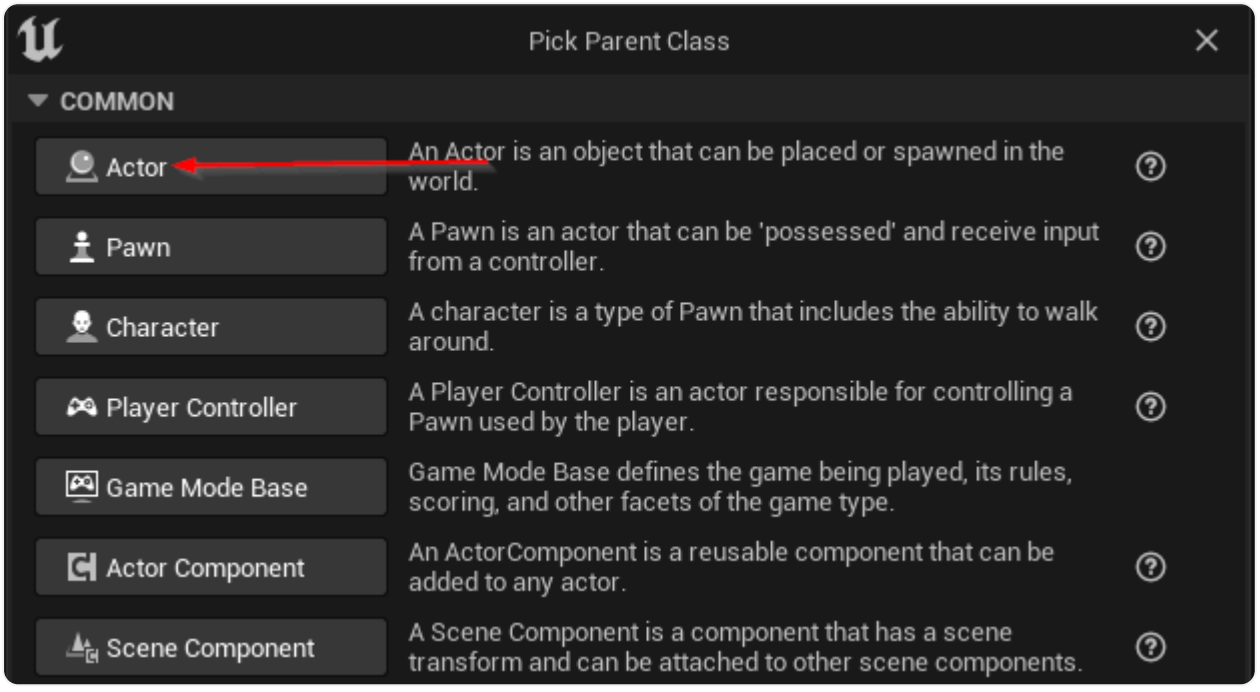
Step 3: Adding a button to your level

In this step we will describe how to create your own button, if you want to use a default button created by YOM please use the `YomDefaultPortalButton`, which works by pressing `E` in front of the button and go to step 4.

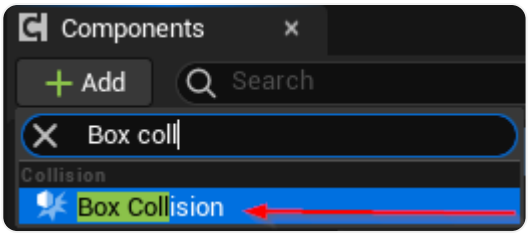
To create your own button we will have to create a blueprint. Right-click in a folder in the `Content Browser` and under `Create Basic Asset` click on `Blueprint Class`.



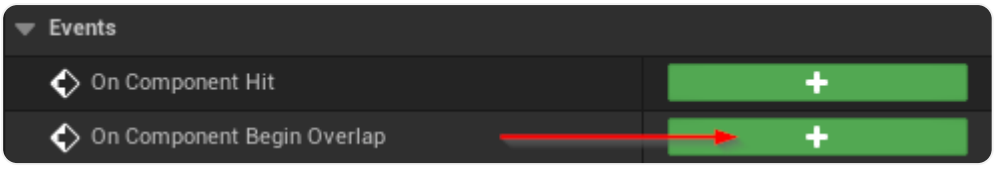
This will open up the blueprint dialog box, in this box click on `Actor`.



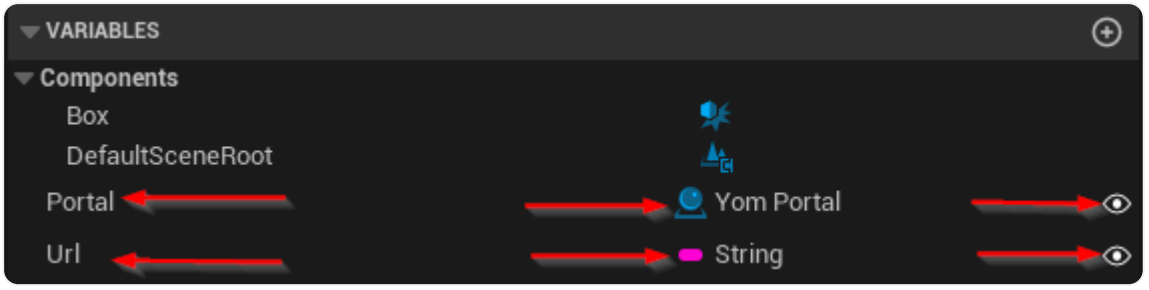
Open up the blueprint and add a collision (we use `Box Collision`).



Click on the new collision component and under `Details -> Events` click on the `+` sign on the `On Component Begin Overlap`.

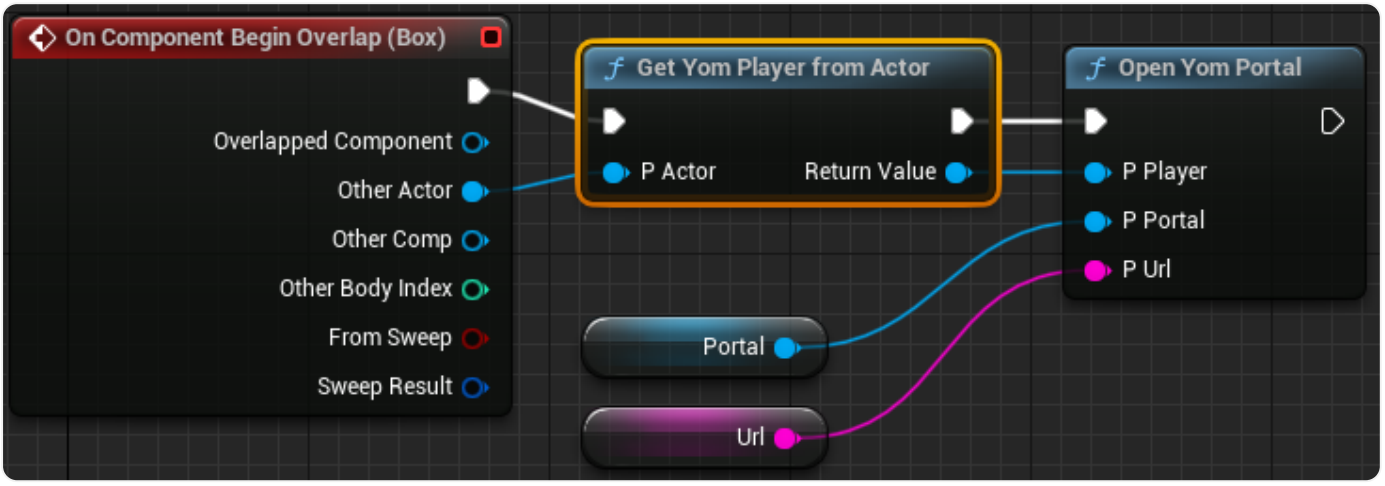


This will now open the event graph. Under `Variables` press the `+` sign and add the following two variables:



Note that you will have to set the types of the variables yourself, and that the 'eye' needs to be open.

After this setup your graph like this:



You have now created a button that can open a portal when a player runs into the box.

Step 4: Adding the button to your level

You can now add the newly created button to your level by taking it from the `Content Browser` and sliding it into your level.

Step 5: Opening the portal

In the `Outliner` find the button that you just put into your level. In the `Details` panel of this actor you can now find the `Portal` and `Url` variables that you created. In the portal set the portal that you added to the level in step 2. The url of the portal should be a valid url to another stream.

Step 6: Checking the images that are generated

If you start the game now, trigger the button and walk over to your portal you will see the image of your stream in the portal. Furthermore, a directory called `PortalImages` is added to your project root in which the images are stored. These are needed when submitting your project to YOM as these will be used to display your stream in other streams. When your level is hosted the image of another stream will be set in here.

unity

> **Coming Q2 2026:** The Unity SDK is in active development, targeting Q2 2026. In the meantime, any Unity game compiled to Linux x86_64 can stream on YOM today using the Quick Start guide. No SDK required.

If you'd like early access or have questions about Unity support, contact us at info@yom.net.

SECTION

about

token allocation

Attribute	Value
Total tokens	750,000,000
Smart Contract	0xb6314518b61b4864162c7aE7fdc36261e0A14C4b
Home Chain	Avalanche (C-Chain / YOM L1)
Trading Chains	Avalanche C-Chain
TGE Price	$0.100
Initial market cap	$13,000,000
TGE FDV	$75,000,000

Token Allocation

Category	Allocation	Tokens	Unlock Schedule
Treasury	37.50%	281,250,000	20% at TGE, 9 month cliff, 36 months linear
Ecosystem	25.00%	187,500,000	20% at TGE, 6 month cliff, 24 months linear
HODL	17.31%	129,790,869	9 month cliff, 16 months linear
Liquidity Pool	9.72%	72,934,111	35% at TGE, 24 months linear
Seed Sale	2.00%	15,000,000	3 month cliff, 10% unlock at cliff, 6 months linear
Private Sale	1.50%	11,250,000	3 month cliff, 12 months linear vesting
KOL Round	0.48%	3,571,429	20% at TGE, 3 month cliff, 6 months linear
Team	3.00%	22,500,000	9 month cliff, 20 months linear
Community	2.78%	20,860,735	3 month cliff, 5 months linear
Exchanges	0.03%	200,000	100% at TGE
Launchpad (i)	0.10%	714,286	10% at TGE, 3 month cliff, 6 months linear
Launchpad (ii)	0.10%	714,286	10% at TGE, 3 month cliff, 6 months linear
Launchpad (iii)	0.10%	714,286	10% at TGE, 3 month cliff, 6 months linear
Quests & Mindshare	0.40%	3,000,000	3 month cliff, 2 months linear

- **Treasury (37.50%)** — Fuels adoption, exchange listings, studio grants, and day-to-day operations. The treasury powers all non-programmatic transactions aimed at accelerating decentralized cloud compute adoption.
- **Ecosystem (25.00%)** — Reserved for node operators. Continuously replenished through network rewards to support idle node time via a flat payout rate — ensuring global coverage from day one.

- **HODL (17.31%)** — Dedicated investors who committed to holding for the very long term. This allocation rewards conviction with an extended vesting schedule.
- **Liquidity Pool (9.72%)** — Dedicated to DEX and exchange liquidity. Ensures deep trading pools across centralized and decentralized venues.
- **Seed Sale (2.00%)** — Institutional equity and token warrant investors from day one, plus seed token investors. 3-month cliff, 10% unlocks at cliff end, with the remainder vesting linearly over 6 months.
- **Private Sale (1.50%)** — Private token investors on extended vesting. 3-month cliff with no unlock at cliff end; full balance vests linearly over 12 months.
- **KOL Round (0.48%)** — Key Opinion Leader campaign for influencer and KOL partners contracted for token-launch marketing reach. 20% unlocks at TGE for aligned launch promotion; the remaining 80% vests linearly over 6 months after a 3-month cliff.
- **Team (3.00%)** — Allocated across all core team members. Vests over a long horizon to ensure deep alignment with the network's success.
- **Community (2.78%)** — Community investors who believed in the project early on. This pool represents our OG holders who chose to hold and bridge.
- **Exchanges (0.03%)** — Reserved residual for launch partners and exchange listings.
- **Launchpad (i / ii / iii) (0.10% each, 0.30% total)** — Three independent community launchpad tranches. Each tranche has a $50k hardcap at $0.07 (a 30% discount to the $0.10 TGE listing price, round valuation $52.5M vs $75M TGE FDV). Vests 10% at TGE, then 3-month cliff and 6-month linear vest.
- **Quests & Mindshare (0.40%)** — Powers leaderboard competitions and influencer partnerships. Supports marketing collaborations that drive awareness and adoption.

roadmap

YOM is building the decentralized future of cloud gaming — one milestone at a time. We believe in shipping a working product first and decentralizing incrementally, so operators earn from day one while the network evolves toward full decentralization.

Here's where we stand and what's coming.

What's Live Today

These features are shipped and operational across the YOM network:

Streaming & Gaming

- Instant browser play — any game, any device, no downloads
- Universal streamer with WebRTC video/audio and full input support (mouse, keyboard, gamepad, mic)
- Cross-platform streaming across all hardware configurations
- Game developer SDK for Unreal Engine 5 with responsive HUDs, events, and Web3 features

Node Network

- NANO plug-and-play devices — boot from USB, scan QR, start earning
- Over-the-air updates for all NANO devices in the field
- Per-session payouts to node operators
- Node reputation (XP) tracking based on uptime and reliability
- Hardware rig scanning and automated verification
- Multi-instance scaling — run multiple concurrent game sessions on a single rig
- Comprehensive node and session logging and monitoring

Platform

- V2 app.yom.net — redesigned operator portal
- V2 stats.yom.net — public network statistics dashboard
- V2 admin portal for internal operations
- Referral program with commission tracking
- Analytics integration (Snag & Mindshare data)
- Advertising integration with partner networks

Security

- Third-party security audit completed
- Hardened Docker containers with vulnerability scanning
- Secured backend API (rate limiting, auth hardening, input validation)
- Encrypted database layer with access controls and audit logging
- LUKS2 full-disk encryption with Tang network unlock on all nodes

---

In Progress

Actively being built and rolling out:

Smarter Network

- **AI-powered resource allocation** — ML models for intelligent node matching and predictive network optimization
- **Push-based architecture** — migrating from polling to real-time data, reducing latency and server load

Scaling Infrastructure

- **Load balancing** — WebSocket server and Kafka cluster balancing for 10x network growth
- **Build upload pipeline** — automated CI/CD for faster game onboarding
- **Containerized nodes research** — exploring Docker/K8s deployment for cloud-based node operators

Blockchain & Tokenomics

- **$YOM Token Generation Event** — token contracts are live on Avalanche C-Chain; TGE date is paused while we complete MiCA statutory review, DTI / LEI registrations, and Tier-1 exchange compliance. Date will be announced once those clear — see our TGE update and strategic update for context
- **Per-session reward distribution** — refined payout calculation tied to token generation event
- **Automated vesting** — smart contract-based token distribution schedules
- **NFT migration to Avalanche** — moving node license NFTs from peaq to Avalanche for better liquidity and lower fees

---

Coming Next

Active development, sequenced as the network and TGE conditions clear:

For Node Operators

- **AMD GPU support** — extending beyond NVIDIA to double the addressable hardware market
- **Delegation support** — delegate your node license to a professional provider and earn passively without running hardware. In active development; partners under discussion
- **In-app license purchasing** — buy node licenses directly from the operator portal
- **Workload difficulty multiplier** — higher payouts for more demanding games (resolution, FPS, graphics)

For the Network

- **Avalanche L1 migration** — custom chain optimized for gaming microtransactions with sub-second settlement, triggered when C-Chain gas costs exceed 4% of session payout
- **Regional tokens** — localized pricing tokens ($yEU, $yCHN) paired against $YOM for region-specific demand signals
- **Burn mechanism** — deflationary pressure through per-session token burns
- **HyperOrch AI scheduler** — ML-driven node matching that learns from session quality and predicts optimal placement

For the Community

- **YOM DAO** — decentralized governance for treasury management, protocol upgrades, and community-driven priorities
- **USD-pegged payouts** — oracle-based conversion for stable operator earnings regardless of token volatility
- **Unity SDK** — game engine support beyond Unreal Engine 5

Our Approach

YOM's roadmap follows three principles:

1. **Ship first, decentralize incrementally** — a working product generates real revenue for operators today while we progressively replace centralized components with decentralized alternatives.
2. **Operators earn from day one** — we don't gate earnings behind unfinished milestones. If you plug in a NANO, you start earning immediately.
3. **Transparency over hype** — this roadmap reflects real engineering priorities. When something is "coming next" it means we're actively planning it, not just talking about it.

Last updated: May 2026

social media

YOM is active on different types of social media platforms.

You will find different types of content and communities throughout our social channels. We invite you to join us in the conversation about Your Open Metaverse, the Metaverse itself, and related topics.

- Twitter
- Telegram Community
- Telegram Announcements
- Discord
- YouTube
- Medium
- Tiktok
- LinkedIn
- LinkedIn Group
- Instagram

faq

1. General Questions

What is YOM?

YOM builds the fastest and most affordable GPU infrastructure on the planet: a decentralized edge network that turns idle consumer GPUs into a global compute grid. Gaming is the first workload because it is the hardest real-time one; the same network already serves enterprise interactive 3D and is built for low-latency AI inference next. YOM connects idle GPU resources from self-hosted and delegated nodes to studios, brands, and broadcasters at a fraction of centralized cloud cost.

How does YOM work?

YOM utilizes a decentralized network of gaming machines (nodes) running our custom **Node OS**, which stream gaming sessions to end users. Studios and brands pay for these services, and node operators earn rewards in $YOM tokens for providing computing resources.

Who can benefit from YOM?

- **Node Operators:** Earn passive income by contributing GPU resources to the network.
- **Community Members:** Participate in governance, earn XP rewards, and engage with the ecosystem.
- **Studios & Brands:** Leverage YOM to provide immersive, high-quality experiences to their audiences at a fraction of traditional cloud gaming costs.

---

2. Node Operators

How can I start running a YOM node?

You can start by choosing one of two options:

1. **Delegation (Hands-off approach):**
   - Purchase a node license for $249 and delegate it to a Node-as-a-Service (NaaS) provider.
2. **Self-Hosting (Hands-on approach):**
   - Purchase a **plug-and-play device** or set up the node via a **manual code installation** on your gaming rig.

What are the hardware requirements for self-hosting?

- **GPU:** Any NVIDIA RTX config (2060RTX and higher) with high VRAM to maximize node capacity.
- **CPU:** Intel i5/i7 or AMD Ryzen 7/9 for better multitasking and stability.
- **RAM:** 16GB would be sufficient on low config setups, but 32GB+ is recommended.
- **Storage:** No storage needed for plug n play solution. For custom Linux boot deployment: 1TB+ available space needed.
- **Internet:** Wired Ethernet connection (min 25mbps upload).

How much can I earn by running a node?

Earnings depend on workload demand and utilization. On average:

- **Per node:** ~$58/month at moderate utilization.

- **Full host (up to 8 licenses):** up to ~$466/month.
- **Payback:** roughly 6 months on a single node.

A NANO is $349 (one license included); additional licenses are $249 each. Rewards are settled by the protocol in $YOM and scale with the sessions your node serves.

What is the difference between delegation and self-hosting?

- **Delegation:** Fully managed by a NaaS provider, with minimal effort required. Lower technical involvement but lower optimization potential.
- **Self-Hosting:** Provides full control over operations and the potential for higher earnings with greater responsibility for setup and maintenance.

What costs should I consider when self-hosting?

Costs include:

- **Hardware acquisition:** New rigs vs. using existing setups.
- **Electricity costs:** Managing power efficiency for profitability.
- **Scaling costs:** Adding more node licenses to increase earnings.

---

3. Studios & Brands

Why should I use YOM for cloud gaming?

YOM offers:

- **Cost Efficiency:** Pay infrastructure cost only, ~$0.13/hr blended versus the ~$2/hr floor of centralized cloud.
- **Scalability:** Access a global network of nodes without infrastructure investment.
- **Ease of Integration:** Embed streams directly into websites, apps, and platforms with a single line of code.
- **Customization:** Implement branding, analytics, and monetization strategies easily.

How is game performance ensured across different devices?

YOM uses an **AI-powered Resource Orchestrator** that dynamically selects the best nodes based on:

- **Latency:** Ensuring the lowest possible ping.
- **Node Reputation:** Prioritizing high-performing nodes with good uptime.
- **Game Availability:** Allocating nodes with pre-loaded game assets for faster deployment.

What pricing model does YOM use?

YOM follows a **session-based pricing model**, with a fixed per-session fee by workload tier:

- **$0.02 / $0.05 / $0.10 / $0.15 per session** for Light / Standard / Performance / Ultra workloads (~$0.13/hr blended).
- **Kingmaker Self-Service Hosting at $89 per slot per month** (100 sessions included; overage billed at the tier rate plus a 20% markup).

Can sessions be ad-supported?

Playable ads and ad-supported sessions are on the roadmap rather than live today. The same embeddable streaming that powers free demos (Discovery) can carry pre-session or playable-ad formats, and we will expand ad-supported models once network density supports it. Until then, monetization runs through subscriptions (the Compute Pass) and publisher hosting (Kingmaker).

---

4. Tokenomics & Rewards

What is $YOM and how does it work?

$YOM is the native utility token of the YOM ecosystem, used for:

- Paying for cloud gaming services.
- Rewarding node operators for their contribution.
- Providing liquidity on decentralized exchanges to earn rewards.
- Ensuring network security and sustainability through a **deflationary model** (burn on usage).

What is XP and its role in the ecosystem?

XP is a **soul-bound value** that represents user contributions and engagement in the ecosystem. It is used to:

- Boost leaderboard rankings for better workload allocation.
- Weight network governance as on-chain DAO voting rolls out (targeted 2027); until then the protocol is stewarded by the YOM Foundation.
- Gain eligibility for additional $YOM drops and rewards.

Is $YOM inflationary or deflationary?

YOM follows a **deflationary model**, meaning no new tokens are minted, and a portion of each transaction is burned, reducing supply over time. This ensures value retention for stakeholders.

How can I earn more rewards?

Increase earnings by:

- Maintaining high uptime and reliability for better workload allocation.
- Participating in the **Streak Program**, which provides bonus rewards for consistent online presence.
- Accumulating XP to boost node rankings and priority.

---

5. Governance & Community

How does governance work in the YOM ecosystem?

YOM is moving toward a **DAO model** (see roadmap), which will allow the community to influence key decisions, such as:

- Platform policies and content guidelines.
- Allocation of treasury funds to support network growth.
- Voting on feature updates and ecosystem improvements.

Who governs the protocol today, and how does that change?

Today the protocol and treasury are stewarded by the **YOM Foundation**, an independent non-profit entity, with professional market makers contracted for token liquidity. As on-chain DAO governance activates (targeted 2027), voting on protocol parameters opens to the community, with weight blending XP reputation (~67%) and $YOM holdings (~33%) so the operators doing the work help steer the network.

How can I participate in the YOM community?

You can engage with YOM through:

- Joining discussions in **Discord and Telegram.**
- Contributing to development and governance through XP-based voting.
- Running community events to onboard new users and expand the ecosystem.

---

6. Technical & Troubleshooting

What should I do if my node is not appearing in the dashboard?

- Ensure your wallet is properly connected.
- Verify your internet connection and reboot the system.
- Contact YOM support if the issue persists.

How do I optimize my rig for better performance?

- Ensure proper cooling and power efficiency.
- Use high-speed wired internet connections.
- Regularly monitor performance metrics through the dashboard.