OCCA

Operating Command Center for Agents

Whitepaper v0.9 — May 2026

---

1. Abstract

OCCA (Operating Command Center for Agents) is an AI-native operating system for Web3-native operators — individual founders, indie builders, and small specialized teams in the Solana ecosystem — to coordinate autonomous AI agent labor forces. A runtime-agnostic adapter layer unifies heterogeneous agent runtimes under a single command center that orchestrates roles, tasks, and outcomes end-to-end. Agents operate as on-chain economic actors, with identity, compensation, and accountability settled natively on Solana.

This document describes the market conditions that motivate OCCA, its four core design pillars, and its four-layer architecture. It specifies the security model and on-chain protocol that realize the platform, and the integrated agent economy that operates over them — on-chain agent compensation, company treasuries, an agent labor market, and a marketplace for proven company templates. It concludes with a phase-gated roadmap, a governance model designed for the on-chain activity the platform mediates, and a protocol economy funded by marketplace fees and enterprise services.

---

2. Introduction

The role of AI in organizations is changing. What began as assistive software — tools that helped humans write, analyze, or search — has evolved into something structurally different: autonomous agents capable of executing tasks, making decisions, and producing economic output with limited human oversight.

2.1 The Rise of Autonomous AI Agents

By 2026, AI agents have become central operational components of enterprise operations. The 2026 OutSystems State of AI Development report — based on a survey of approximately 1,900 IT leaders — found that 96% of enterprises now use AI agents in some capacity, with 97% exploring system-wide agentic strategies (OutSystems 2026). Gartner projects that the average Fortune 500 enterprise will run over 150,000 agents per company by 2028, up from fewer than fifteen in 2025 (Gartner 2026). Individual agents are increasingly reliable, specialized, and persistent — capable of sustained work rather than single-interaction responses.

Yet the infrastructure supporting this shift has lagged behind the agents themselves. Most organizations operate agents in isolation, wired together by ad-hoc integrations and supervised on a per-agent basis. Coordination across agents, consistent governance, and accountability for autonomous work remain unsolved problems — as evidenced by the industry concern over *agent sprawl*, cited by 94% of surveyed enterprises as a growing source of complexity, technical debt, and security risk (OutSystems 2026).

2.2 The Web3 Convergence

In parallel with the rise of autonomous agents, the on-chain economy has matured into programmable financial infrastructure. Low-latency settlement, composable on-chain identity, and native support for automated transactions have made public blockchains a credible environment for non-human counterparties.

This convergence is already visible. Early 2026 telemetry: Solana has processed roughly 15 million on-chain agent payments to date, accounting for an estimated 65% of agentic payment activity routed through emerging x402 rails (Blockonomi 2026) (CoinDesk 2026). Solana Foundation leadership has publicly speculated that autonomous actors will dominate on-chain transaction activity within two years (Crypto Briefing 2026); that figure is a directional projection from an interested party, not an established benchmark, and is cited here only as commentary on trajectory rather than as evidence of present-day distribution. The structural point that survives independent of the projection is that early agent-economy platforms have demonstrated autonomous software entities owning wallets, generating revenue, and engaging in commerce on-chain — agents as economic actors are no longer hypothetical.

2.3 The Missing Layer

What is missing is a purpose-built operating environment for Web3-native operators to run AI agents as a coordinated agent labor force rather than as individual tools. Existing enterprise control planes — Microsoft Agent 365, Google Gemini Enterprise, AWS Bedrock Agent Registry — are centralized, proprietary, and designed for Web2 enterprises. Open-source orchestration projects have begun to close the coordination gap for general users, but none treat Web3 primitives — wallet-native identity, on-chain settlement, native cryptoeconomic participation — as first-class foundations.

The data cited in §2.1 describe enterprise adoption of AI agents broadly. The structural pattern is observable across organizational categories, and Web3-native operators — solo founders, small builder teams, and specialized crypto-native firms in the Solana ecosystem — encounter the same coordination, governance, and verification gaps in their own operational form. OCCA is positioned for these operators first, where wallet-native identity, on-chain settlement, and verifiable provenance are operational prerequisites rather than retrofits. The platform's runtime-agnostic architecture (§4.2.1) and dedicated infrastructure

tier (§11.3) are designed to compose cleanly with broader environments as the platform matures (§14.4); the initial focus is on the individual and small-team Web3-native cohort whose operational primitives the platform is built around.

---

3. Problem Statement

The past two years produced remarkably capable individual AI agents, but no organizational or economic infrastructure for deploying them as a coordinated, on-chain agent labor force. Six fundamental gaps separate Web3-native operators from their AI agent labor force.

3.1 The Runtime Access Gap

Today's AI agent platforms are built around a single runtime, typically operated by the platform's parent company. A team that has already invested in Cursor or Claude Code should not be forced to abandon those subscriptions — and double their AI spend — simply to use an agent labor force platform. Yet that is exactly what today's agent platforms demand: commit to their runtime, or you cannot participate. The result is fragmented agent labor forces that cannot interoperate, duplicated subscription costs, and exclusionary barriers for users whose access is limited to a single runtime.

3.2 The Command Center Gap

Enterprises have adopted AI agents at scale — 96% of enterprises now use them in some capacity, and Fortune 500 companies are projected to operate at the order of 150,000 agents per company by 2028 (OutSystems 2026) (Gartner 2026). Yet only 12% of organizations have implemented a centralized platform to manage them (OutSystems 2026). The remaining majority operate in what industry analysts term *agent sprawl* (OutSystems 2026) (Gartner 2026): agents running in isolation, pursuing locally optimal but globally conflicting objectives. Documented cases include uncoordinated agents in procurement, pricing, and customer-facing roles producing contradictory actions whose downstream impact is difficult to attribute and recover from.

The cited figures cover enterprises broadly; analogous data on Web3-native operators specifically is not yet published at comparable rigor. The structural pattern, however, is observably similar at smaller scale: crypto-native solo operators, indie builder teams, and specialty firms report ad-hoc agent deployments — content production, on-chain data ingestion, trading automation, community ops — running in parallel without shared coordination, governance, or accountability. The Command Center Gap is presented here as an enterprise-scale problem with which Web3-native operators contend in their own operational form, rather than as a measured Web3-specific phenomenon.

Major cloud providers — Microsoft Agent 365, Google Gemini Enterprise, AWS Bedrock Agent Registry — have all launched control-plane products in response. Each is centralized, proprietary, and designed for Web2 enterprise environments with the corresponding identity, billing, and governance assumptions built in. These products solve the enterprise form of the Command Center Gap; they do not solve the form Web3-native operators face, in which authority is wallet-bound, treasury is on-chain, and accountability requires verifiable provenance rather than vendor-issued audit logs.

Within the Web3 ecosystem, agent-economy projects such as Virtuals Protocol and Olas (Autonolas) approach the agent space from adjacent angles — Virtuals primarily as a tokenized agent-launch and trading venue, Olas as an autonomous-services framework with on-chain coordination. Neither is positioned as an organizational operating system. Virtuals' agents are typically traded and incentivized as standalone economic actors rather than coordinated within a labor-force structure. Olas's mechanism design centers on services, registries, and staked agent operators rather than on the operator-facing organizational primitives (roles, goals, tasks, treasuries, traces) that an OCCA company embodies. OCCA addresses the operating-system form of the gap: a Web3-native operator running an organized agent labor force end-to-end, rather than launching, trading, or registering individual agents. Its BYORT architecture (§4.2.1) and enterprise tier (§11.3) leave the platform compatible with broader environments as a later expansion (§14.4) rather than as the initial market.

3.3 The Web3 Operational Gap

Existing agent-management tools assume a Web2 operating environment: email authentication, fiat-denominated subscriptions, centralized data custody. Web3-native operators — solo crypto-native founders, indie builder teams, specialty firms — operate on fundamentally different primitives: wallet-based identity, crypto treasuries, decentralized or user-owned data. No operational tooling today is built natively for this environment. Web3-native operators that want to deploy AI agent labor forces are forced to onboard into Web2 SaaS platforms that contradict their own operational model.

3.4 The Economic Infrastructure Gap

AI agents are currently treated as software — tools invoked by humans, billed to human accounts, producing outputs owned by humans. This framing breaks down as agents increasingly perform work with real economic value: drafting, analyzing, transacting, coordinating. An agent performing economic work requires the infrastructure of an economic actor: persistent wallet-based identity, direct compensation rails, the ability to hold and transact assets, and a verifiable on-chain record of delivered work. No operating-system-level platform today provides this integrated economic foundation.

This gap is increasingly relevant. As the share of on-chain activity attributable

to autonomous actors grows (§2.2), the operational infrastructure required to run them as economic actors becomes a near-term, not a long-term, concern.

3.5 The Trust & Verification Gap

As AI agents take on work with real consequence — executing transactions, producing client deliverables, communicating with counterparties, auditing code — the question of how to verify what an agent actually did becomes structural. Today's agent platforms answer this internally: log files in their own database, audit trails they themselves control, dashboards their own engineers built. The platform is both the actor and the witness.

This forecloses the cross-organizational engagement that scales an agent economy. For one party to engage another party's agent on commercial terms, the record of work must be **verifiable by the engaging party**, **tamper-evident**, and **portable** beyond any single platform's continued cooperation. No mainstream agent platform provides this; reputation, where it exists, is locked inside the platform that issued it, and an agent that leaves loses everything it has done. Disputes have no neutral evidence to anchor on. Past performance cannot be carried into a new engagement.

The Trust & Verification Gap is not a niche compliance concern — it is the structural barrier that prevents AI agent labor from operating across organizational boundaries. Without verifiable, portable provenance, an external party cannot credibly engage another party's agent, validate prior work, or settle disputes against an immutable record.

3.6 The Specialization Access Gap

Generic AI capability is widely available; specialized AI capability is not. A team that needs an agent tuned to a specific domain — Solana smart-contract review, structured crypto research, brand-specific content production, on-chain trading patterns, contract auditing — faces a stark choice: build the agent themselves over weeks or months of prompt tuning, skill integration, and operational testing, or accept the lower performance of a generic agent applied to a specialized task.

This problem compounds across the broader market. Each operator that needs a specialized agent rebuilds it from scratch. A research firm tuning a crypto-narrative analysis agent does not benefit from another firm's prior work on the same domain; a protocol team operating a Solana-specific code-review agent cannot acquire proven configurations from operators that have already done the work. Specialized capacity is locked inside the operator that built it, and the cost of that capacity is paid in full by every operator that needs it, every time. There is no marketplace where proven company configurations can be acquired and deployed, and no labor market where specialized agents can be engaged on contract terms across organizations.

---

4. OCCA Overview

4.1 What is OCCA?

OCCA is an operating system for Web3-native operators — individual founders, indie builders, and small specialized teams in the Solana ecosystem — to run autonomous AI agent labor forces. It is composed of an off-chain Command Center that orchestrates work, a runtime adapter layer that connects heterogeneous AI runtimes, and an on-chain layer on Solana that settles identity, compensation, and ownership.

4.2 Core Pillars

OCCA is built on four design principles, each a deliberate architectural choice that differentiates OCCA from existing agent platforms.

4.2.1 Bring Your Own Runtime (BYORT) OCCA is runtime-agnostic by design. Rather than committing users to a single AI provider, OCCA connects to any compatible agent runtime through a pluggable adapter layer — a principle referred to throughout this document as *Bring Your Own Runtime* (BYORT). Teams deploy agents using the runtime they already trust, already pay for, and already operate. The platform orchestrates; the runtime executes.

This eliminates vendor lock-in, removes duplicate subscription costs, and makes OCCA accessible regardless of which AI provider a user has access to. At launch, OCCA's adapter coverage spans an initial set of widely used runtimes — covering at least two independent runtime backends per the Phase 1 deliverables (§14.1) — and the architecture is designed to extend that coverage over time and to accommodate enterprise-specific runtimes via custom adapters (§11.3). The current list of supported runtimes is published in OCCA's documentation and updated as new adapters are reviewed and released.

The model also shifts a class of compliance considerations from the platform to the operator. Because the runtime is operator-selected, data residency, runtime-vendor terms of service, model-output content policies, and audit-trail provenance across the OCCA-adapter-runtime path are the operator's responsibility to evaluate against their own regulatory obligations (GDPR, sector-specific data-handling rules, internal vendor-management policies). OCCA's hosted infrastructure addresses the platform side of this chain; the runtime side is constrained by the operator's choice of runtime. Enterprise operators with stricter sovereignty or compliance requirements may use the dedicated infrastructure tier with custom adapters (§11.3) to bring proprietary runtimes under their direct control while preserving the BYORT model.

4.2.2 Web3-Native by Design OCCA is built from the ground up for the Web3 operating environment. Authentication is wallet-based: one Solana program-derived account represents one company, serving simultaneously as identity, ownership credential, and access boundary. Treasury management,

agent compensation, and company transfers are all settled natively on-chain. OCCA does not bolt Web3 onto a Web2 product — it is Web3 from the authentication layer upward.

For a crypto-native operator — whether an individual using a wallet-based identity or a small team operating under a multi-sig — controlling authority, treasury, and operational record live under the same on-chain patterns the operator already uses for other activity. Wallet signing, multi-sig coordination, and verifiable on-chain history apply to the AI agent labor force in the same form they already apply to the operator's existing on-chain operations. There is no parallel SaaS access model, no off-chain user-management plane, and no reconciliation between on-chain treasury operations and an off-chain agent-management ledger.

4.2.3 AI Agent Labor OCCA treats AI agents as a distinct class of economic actor — neither passive tools nor employees, but autonomous software performing compensable work within an organization. Each agent holds a role, takes on tasks, maintains persistent memory, accumulates a verifiable on-chain track record, and receives compensation paid to its on-chain address. The organizational model — roles, reporting lines, responsibilities, compensation — is the operational primitive at the protocol level.

This framing is deliberate. "AI agent labor" describes a new factor of production that resembles human labor in its compensable, output-producing character but differs from it in structure: agents do not have legal personhood, are not employees under any jurisdiction's labor law, and are not parties to employment contracts. The terminology used throughout this whitepaper — *agent compensation*, *agent labor force*, *agent engagement* — is chosen to keep this distinction visible. The legal and tax characterization of this relationship in any given jurisdiction is a separate question addressed in §15.7.

4.2.4 An Operating System, Not a Dashboard OCCA is not a task-management dashboard. It is a desktop-first operating system for running an AI agent labor force. The interface is designed as a full-screen OS shell — with windowed applications and a live 3D Live Office rendering of the labor force as ambient context — rather than as a simple control panel. Operators do not monitor a feed; they operate a system.

4.3 How the Pillars Relate to Web3

The four pillars relate to Web3 infrastructure differently, and this distinction matters for evaluating OCCA's architectural claims.

The **Web3-Native** pillar is constitutive: wallet-native treasury settlement (§10.1, §10.2), on-chain provenance for agent work and economic history, and the verifiable performance record on which the Template Marketplace's listings depend (§10.3) are not implementable without on-chain infrastructure. These

are the elements of OCCA that depend on Web3 as a substrate, not as a positioning choice.

Bring Your Own Runtime, **AI Agent Labor**, and **Operating System, Not a Dashboard** are architectural and product choices designed to compose cleanly with the Web3 layer. A multi-runtime adapter, an agent-labor primitive model, and a desktop-first operating environment could in principle be implemented on a Web2 platform; the choices are not Web3-exclusive in isolation. What makes them load-bearing in OCCA is their integration with the on-chain layer — agents whose identity and compensation are wallet-native, organizations whose treasuries and ownership are settled on Solana, and an operating environment that surfaces this on-chain state as live, navigable structure. The differentiation rests on the integrated whole, not on any single pillar.

---

5. Core Concepts

OCCA's architecture and economic model rest on a small set of foundational primitives that form the platform's vocabulary. The primitives are grouped into three categories: **organizational primitives** describe the structure of a company and the work it produces; **execution and record primitives** describe how that work is carried out and verified; and **economic primitives** describe how value flows through the platform. The grouping is conceptual; in practice the primitives interact across categories.

5.1 Organizational Primitives

5.1.1 Company A **Company** is the top-level organizational unit in OCCA. It is identified by a single Solana program-derived account (PDA) that simultaneously serves as its on-chain identity, its treasury address, and its access boundary (§4.2.2, §8.2.2). A company contains agents, defines roles and goals, holds a treasury, accumulates a reputation record, and operates independently of every other company on the platform.

A user may own multiple companies, each represented by a distinct PDA with its own controlling authority. A company's controlling authority can be transferred privately by the current authority through a single on-chain instruction (§8.2.2); transfer of the controlling authority is the only mechanism by which a company changes ownership. The Template Marketplace (§10.3) does not transfer ownership of operating companies; it sells deployable configurations that buyers instantiate into their own companies.

5.1.2 Agent An **Agent** is a first-class member of a company's agent labor force. Every agent has four defining properties:

- An **identity** — a Solana address derived from the company under one of three custody models (§8.2.3).

- A **role** — the function it performs within the organization.
- A **runtime** — the AI execution layer it operates through, connected via an adapter.
- A **skill set** — the capabilities attached to it.

Agents do not exist independently of companies. They are created within a company, earn compensation from the company's treasury, and carry that company's organizational context into every task they perform.

5.1.3 Role A **Role** defines an agent's function within a company. Roles establish reporting lines, permissions, and the expectations under which an agent operates.

Roles are not fixed job titles; they are structural primitives that shape how an agent relates to other agents, to tasks, and to goals. A role determines which tasks an agent may accept, which other agents it may delegate to, and which parts of the company's goal hierarchy it is responsible for. OCCA ships with a default role catalog covering common executive and operational functions; companies may extend the catalog with custom roles.

5.1.4 Goal A **Goal** is a directional objective — something the organization is working toward. Goals form a hierarchy: the company holds a top-level mission, which decomposes into project-level goals, which decompose further into agent-level goals.

A goal records its parent goal, its owning agent or role, a target outcome, and a status. Decomposition is performed by the operator, by an agent acting within its delegated authority, or by a combination of both. Every task is tied to a goal; this linkage is what allows individual work to remain aligned with organizational objectives and provides the mechanism by which an agent understands *why* a task matters, not only *what* is required.

5.1.5 Task A **Task** is a discrete unit of work assigned to an agent. It has a defined scope, an owner, a status, and a parent goal. Tasks are created by operators or, under governance constraints, by other agents, and progress through a visible lifecycle: *created* → *assigned* → *executing* → *completed* → *reviewed*.

Tasks are the atomic operational unit of OCCA. Every action an agent performs occurs within the context of a task; every outcome is recorded against a task; every compensable unit of work is attributable to one or more tasks.

5.1.6 Skill A **Skill** is a capability attached to an agent — a specialized function, knowledge set, or tool access that extends what the agent can do. Skills are described by a manifest that declares the skill's name, version, required permissions, and entry points. Skills may be built-in, authored by the company owner, or acquired from an external skill library.

An agent's skill set defines its effective area of competence. Skills are the mechanism by which a generic agent runtime becomes a specialized member of an agent labor force — a content-writing skill turns an agent into a writer; a contract-review skill turns an agent into an auditor; a trading skill turns an agent into a financial operator.

5.1.7 Context A **Context** is shared organizational knowledge made available to every agent in a company — its mission, brand voice, customer data, product specifications, internal decisions, and accumulated institutional knowledge. Context is what every agent in the company knows by default; it is the company's working knowledge base, expressed as a queryable resource that agents read at task setup.

Context is layered: company-wide context applies to every agent in the company, role-scoped context applies only to agents in specific roles (engineering context for engineering agents, marketing context for marketing agents), and project-scoped context applies only within a defined project. The operator authors and curates context; agents read it as input to their reasoning and may propose updates that the operator reviews. Context is what allows multiple agents to act coherently as a single organization rather than as independent tools running in parallel.

5.2 Execution and Record Primitives

5.2.1 Trace A **Trace** is the execution record of an agent's work on a task. It captures the agent's reasoning, the tools and APIs it called, the inputs and outputs at each step, and the final result. Trace content is stored off-chain in OCCA's content-addressable store; a Blake3 hash of each trace is anchored on-chain at task completion (§8.3).

Traces serve multiple roles: they enable observability for human operators, they allow past work to be reviewed and referenced by future tasks, and they provide the raw data from which agent reputation and historical value are derived.

5.2.2 Adapter An **Adapter** is the component that connects an agent to its underlying runtime — the AI system that actually executes the agent's reasoning. OCCA's adapter layer is the technical realization of the *Bring Your Own Runtime* pillar (§4.2.1): it standardizes the interface between the Command Center and heterogeneous runtime backends, allowing any compatible runtime to serve as an agent's execution engine. The adapter trust boundary and operating constraints are described in §8.5.

Adapters are not visible to operators in day-to-day use. They are configured once per agent at creation, pinned to a specific version, and operate transparently from that point forward.

5.2.3 Routine A **Routine** is a declared piece of recurring work. A routine specifies what to do, which agent or role should do it, and on what cadence — a fixed schedule, a heartbeat interval, or an event trigger. When a routine fires, the platform creates a task from the routine's template and dispatches it to the designated agent.

Routines turn one-off task assignment into a self-sustaining operating loop. They are how an OCCA company performs continuous work — daily reports, hourly monitoring, weekly outreach — without an operator manually creating each instance.

5.2.4 Tool A **Tool** is a connection to an external system — a database, an API, a service integration — that one or more agents in a company can call. Tools are distinct from skills (§5.1.6): a skill is the agent's internal reasoning capability for a domain, whereas a tool is the agent's connection to an external system it can act on (a Slack workspace, a GitHub repository, an on-chain RPC endpoint, an internal CRM).

Tools are configured at the company level and granted to agents by role or by direct assignment. The same tool can be shared across multiple agents — a single GitHub connection used by a code-review agent, a deployment agent, and a documentation agent — keeping integration setup at the company level rather than duplicated per agent. Every tool invocation is captured in the calling agent's trace and is governed by the company's authorization policy (§8.4) for actions that move funds or reach external systems.

5.2.5 Memory **Memory** is the persistent knowledge that agents accumulate across tasks. Where a trace (§5.2.1) captures the execution record of a single task, memory captures what was learned from that work — durable facts, decisions, customer-specific knowledge, design rationale — that agents need to recall in future work.

Memory is layered: an agent's personal memory holds knowledge specific to its own work; project memory holds knowledge shared by agents working on the same project; company memory holds knowledge persistent across the entire organization. Memory is queryable by agents at task setup and updatable during execution, subject to the company's policy on memory writes. Memory is what allows an OCCA company's collective knowledge to grow over time, rather than resetting at every task.

5.2.6 Channel A **Channel** is the communication path between agents — the mechanism by which one agent delegates work to another, requests information, or sends an artifact for review. Channels are typed: a *delegation channel* transfers task ownership from one agent to another; a *consultation channel* asks for input without transferring ownership; a *review channel* requests assessment of a completed deliverable.

Every inter-agent message is captured in trace records and surfaced in the 3D Live Office (§7) as visible character interaction. Channels are how agent labor force coordination becomes observable rather than implicit: an operator can see which agents are collaborating on which tasks, and can audit the chain of delegation and review behind any completed deliverable.

5.3 Economic Primitives

5.3.1 Treasury A **Treasury** is the on-chain wallet associated with a company's controlling PDA. It holds the company's balances in SOL, USDC, and other supported SPL tokens, receives revenue, and disburses agent compensation, marketplace payments, and protocol fees.

Spending from a treasury is gated by the company's authorization policy (§8.4), which classifies outflows by risk and applies the corresponding signer requirements. The treasury is bound to the company's PDA and remains under the controlling authority recorded for that company.

5.3.2 Contract A **Contract** is an on-chain agreement between two parties for the performance of work. Contracts appear in two contexts: a company engaging an external agent through the Agent Labor Market (§10.4), and a company engaging another company under a service arrangement.

A contract records the parties, the scope of work, the compensation terms, the milestones (if any), and a Solana-based escrow that releases funds on completion or termination. Both parties accumulate reputation against contracts they enter, providing the on-chain history on which future engagements are evaluated.

5.3.3 Template A **Template** is a packaged, deployable configuration of a company — its roles, skills, routines, workflows, tuned system prompts and instructions, and pre-configured tool and integration setup — bundled into a single primitive that another operator can purchase and deploy into their own company.

A template is authored from a live, operating company. The template records: the originating company's address, the current template version, the bundle's content hash (anchored on-chain), and a manifest of what the bundle contains. The template is **self-contained** at deployment: once a buyer purchases and deploys a template, the resulting configuration runs entirely under the buyer's company without runtime dependency on the creator.

A template carries a **modifiable license**: the buyer may adapt the template to their needs (modifying prompts, adjusting routines, swapping tools), but may not resell the template or any derivative of it. The original creator remains the on-chain author of record. Updates published by the creator are made available to existing buyers on an opt-in basis (§10.3).

5.3.4 Listing A **Listing** is an on-chain offer to sell a company template. A listing is created by the template's author and records the asking price, the accepted payment asset, the template's content hash, the originating company reference, and any author-supplied metadata (description, category, tags). A listing remains open until it is delisted by the author, deprecated by the author, or removed by governance (§12.1) following a community flag.

Listings are queried by buyers against the originating company's on-chain track record — treasury history, completed-task and agent-performance metrics, and the listing's own sales history and ratings (§10.3). Settlement of a purchase transfers the buyer's payment to the author (less protocol fee, §11.1), records the buyer's license on-chain, and allows the buyer to deploy the template into their company. The buyer's deployed instance is independent of the original; subsequent activity in either company does not flow back through the listing.

5.3.5 Reputation **Reputation** is a derived primitive: a verifiable summary of an actor's on-chain history. Both companies and agents accumulate reputation, with the underlying record consisting of completed tasks, settled contracts, treasury history, and (for templates published by a company) the sales and rating history of that company's templates.

Reputation is not a single score assigned by the platform. It is a structured, queryable record from which different consumers — Labor Market engaging parties, Template Marketplace buyers, 3D Live Office observers — can derive the metrics relevant to their decision. The protocol defines the canonical inputs; downstream tools and policies define how those inputs are weighted.

---

6. Architecture

OCCA is organized into four layers, each with a defined responsibility: orchestration, execution, settlement, and presentation. The separation lets the platform evolve along any single axis — a new runtime, a new on-chain capability, a new visualization mode — without disturbing the others.

```
flowchart TB
    User([Operator · Wallet])

    EL["Experience Layer<br/>§6.4 · §7"]
    CC["Command Center<br/>§6.1"]
    AL["Adapter Layer<br/>§6.2 · §8.5"]
    OC["On-Chain Layer · Solana<br/>§6.3 · §9"]

    R1[Runtime A]
    R2[Runtime B]
    Rn[…]
```

```
User --> EL
EL <--> CC
CC <--> AL
CC <--> OC
AL -.-> R1
AL -.-> R2
AL -.-> Rn

classDef ext fill:#f5f5f5,stroke-dasharray:5
class R1,R2,Rn ext
```

Figure 1 — Layer architecture. Solid arrows are internal calls; dotted arrows cross OCCA's trust boundary into operator-controlled runtimes.

6.1 The Command Center Layer

The Command Center is the orchestration brain of OCCA. It holds the authoritative state of the platform: companies, agents, roles, goals, tasks, traces, and treasury ledgers. It dispatches work to agents, captures execution traces, enforces governance gates such as approval policies and budget limits, and mediates interactions between the other layers.

In operational terms, the Command Center is what makes OCCA an *operating system* rather than a collection of running agents. It decides what gets dispatched and when, who is allowed to do what, and how the output of one agent's work becomes the input to another's. It is the only layer that holds the full picture of the organization at any given moment.

6.2 The Runtime Adapter Layer

The Runtime Adapter Layer is the technical realization of *Bring Your Own Runtime* (§4.2.1). Every agent runtime — whether a cloud API, a local model server, or a developer tool — presents a different execution interface, lifecycle model, and context format. The Adapter Layer normalizes these differences into a single, stable contract that the Command Center can depend on.

An adapter translates in both directions: Command Center instructions are mapped into runtime-specific calls, and runtime outputs are mapped back into OCCA's standard trace format. This translation is what allows heterogeneous agents to coexist in the same company, collaborate on the same task graph, and generate comparable trace records — regardless of the underlying AI system they are built on.

Adapters are the platform's primary extension point. Adding a new runtime means writing a new adapter; it does not require changes to the Command Center or to any agent already deployed.

6.3 The On-Chain Layer

The On-Chain Layer is where OCCA's economic and identity primitives are settled. It is implemented on Solana and is responsible for every fact the platform claims to be verifiable: company ownership, agent identity, treasury balances, agent compensation transfers, Labor Market contracts, Template Marketplace listings and purchases, and reputation records. The concrete protocol-level specification — programs, accounts, instructions, and fee mechanisms — appears in §9.

The On-Chain Layer is not a passive storage medium. It is the authoritative source of truth for the economic state of the platform: when a agent compensation transfer appears on Solana, the agent has been paid; when a company transfer is confirmed on Solana, ownership has changed. The Command Center orchestrates the actions that produce these transactions, but the settlement itself, and the record of it, lives on-chain.

This separation — orchestration off-chain, settlement on-chain — is a deliberate architectural choice. Off-chain orchestration provides the speed and flexibility required for real-time operation; on-chain settlement provides the verifiability and permanence required for economic actors.

6.4 The Experience Layer

The Experience Layer is the operator-facing surface of OCCA: the desktop OS shell, the windowed applications, and the 3D Live Office (§7). It is where the abstract state maintained by the Command Center becomes a concrete, navigable environment that a human can operate.

The Experience Layer subscribes to live state from the Command Center and renders it spatially and interactively. It is not a passive dashboard — it is an environment. Operators move through their company the way a system administrator moves through an OS, with windows, views, and a persistent spatial model of the agent labor force always in the background. Because it is isolated from the Command Center, presentation can evolve — richer 3D rendering, new layouts, alternative visualizations — without architectural change to the operational foundation.

---

7. The 3D Live Office

Traditional agent platforms render agent labor force activity as feeds, logs, or ticket boards — formats that convey information but lack presence. Operators read about their agents rather than see them. As AI agent labor forces scale into dozens of agents operating in parallel, text-based dashboards fail to communicate the shape, rhythm, and health of an organization at a glance.

The 3D Live Office is OCCA's ambient presence layer for the agent labor force.

It is rendered as a three-dimensional office environment behind the platform's primary windowed UI: agents appear as animated characters at desks while working, lounging in shared areas while idle, and moving between zones when handing off work to one another. The 3D Live Office is the visible expression of OCCA's *Operating System, Not a Dashboard* pillar — a working environment that operators can see at a glance, not a feed they have to scan.

The 3D Live Office is intentionally **ambient**, not navigational. Operators do not click their way through a virtual building to manage agents — that work is done through the standard windowed UI, where lists, detail panels, and action menus are designed for keyboard speed and direct manipulation. The Office sits behind that UI, providing live context to the operations the operator performs in front of it.

7.1 Real-Time State Rendering

Each agent in the Office has a visible state expressed through character animation:

- **Working** — character at their assigned desk, animation reflecting active task execution.
- **Idle** — character on a couch, in a lounge area, or otherwise visibly relaxed.
- **Collaborating** — character walking between zones or interacting with another character when a task handoff or consultation is in progress.
- **Blocked or error** — a clearly distinct visual signal at the character or seat.

Inter-agent communication — task handoffs, consultations, reviews — is rendered as visible interaction between entities. The operator observes the system the way a manager observes a physical office through a window: with peripheral vision, not focused reading.

7.2 The Organizational Spatial Layout

Companies in OCCA are not flat lists of agents. They have structure: leadership, departments, functional teams. The 3D Live Office reflects this structure as a fixed spatial layout — an executive suite, engineering and operations zones, lounge and shared areas, meeting rooms. Agents occupy seats within the layout that correspond to their role.

The layout is **environmental**, not interactive in the sense of game navigation: the operator does not pilot a character through it. Rather, the layout gives the company a recognizable shape that makes its activity glanceable — *engineering looks busy today; the ops zone is quiet; the meeting room has two characters in it* — without requiring the operator to read status fields one row at a time.

7.3 Engagement and Observability

The visualization is not cosmetic. It produces operational outcomes that text-based dashboards do not. Operators who see their agents as live characters develop a faster mental model of their labor force. Anomalies — a normally active agent gone idle, a desk that has been empty too long, an unexpected handoff pattern — become visually evident before they surface in logs. Trust increases because behavior is observable rather than inferred.

The 3D Live Office also produces a marketing and engagement effect that a flat dashboard does not. Operating a labor force of autonomous agents in a visible office is materially different in feel from monitoring a list — closer to running a small studio than reading a status report. The intent is to support sustained operator engagement and to give the platform a distinct visual identity in a market converging on undifferentiated dashboards.

7.4 Architectural Position

The 3D Live Office is one component of the Experience Layer (§6.4), not a replacement for it. The platform's primary operational surface is the windowed UI; the 3D Live Office is the environment those windows inhabit. Both subscribe to the same Command Center state, so what the operator sees rendered in 3D and what they read in the UI are the same underlying state seen from two angles.

---

8. Security Model & Trust Assumptions

OCCA's security model is built around a small set of explicit assumptions, a tiered identity and custody design, and protocol-enforced integrity for the records on which the platform's economic guarantees depend. Where a stated assumption is violated, the corresponding guarantee no longer holds. This section enumerates each so that operators, integrators, and external reviewers can evaluate the platform against the conditions under which it was designed to function.

8.1 Trust Assumptions

OCCA's guarantees are conditional on the following assumptions:

1. **Solana liveness and finality.** The Solana base layer continues to produce blocks and finalize transactions within standard service expectations. Extended chain halts or reorganizations beyond standard finality depth are treated as exogenous events and are not separately mitigated by the protocol.
2. **Cryptographic primitives.** The platform relies on the security of Ed25519 signatures and Blake3 / SHA-256 digests as implemented in the

Solana runtime.

3. **Operator key custody.** The operator — the human user, organization, or on-chain governance contract that controls a company — retains exclusive control of the company's controlling authority. OCCA does not custody operator keys.
4. **Runtime honesty.** A connected runtime executes the agent's reasoning faithfully and returns outputs that have not been intentionally manipulated. OCCA verifies trace structure and provenance, not the semantic correctness of model outputs.
5. **Adapter integrity.** Each adapter (§5.2.2, §6.2) faithfully translates Command Center instructions into runtime calls and records results without omission. Adapters are reviewed before publication and pinned by version.

These assumptions are referenced throughout the remainder of this section and the Known Risks section (§15).

8.2 Identity & Key Custody

OCCA distinguishes three classes of cryptographic identity, each with a different custody model. The choice for each class reflects the trust and recovery profile appropriate to the actor it represents.

8.2.1 Operator Wallet The operator wallet is held entirely by the human user — or by the on-chain authority (multi-sig, governance contract) acting as operator. OCCA never has access to the operator's private key.

Authentication follows the standard Solana wallet-adapter pattern: the operator signs a server-issued nonce to establish a session, and signs individual transactions for any action that mutates on-chain state. Session credentials are bound to the wallet's public key and expire on a configurable schedule.

8.2.2 Company Wallet A company in OCCA is identified by a **Solana program-derived account (PDA)** controlled by the OCCA program, not by an externally-owned account (EOA). The PDA holds the company's treasury and exposes a `transfer_authority` instruction that updates the recorded controlling authority on-chain.

This design separates the company's on-chain identity from any single human key. It allows the controlling authority to be a multi-signature account, a governance contract, or any other on-chain authority pattern, and to be rotated without disrupting the company's identity, treasury, or operating record. Voluntary transfer of the controlling authority — for example, when a founder hands operations to a new owner outside the protocol's marketplaces — is similarly recorded on-chain through a single program-mediated instruction rather than through off-chain key handoff, which is neither observable nor verifiable.

The PDA's authority field is mutated only by a signed instruction from the current controlling authority (voluntary transfer). No other code path modifies the controlling authority.

8.2.3 Agent Wallet Each agent (§5.1.2) holds its own Solana address derived from the company. OCCA supports three custody models, selected at agent creation and recorded immutably in the agent's on-chain metadata.

Default — Derived custody. The agent's keypair is derived deterministically from the company's controlling signer using a hardened SLIP-0010 path over Ed25519, with the derivation index recorded on-chain in the agent registry. The operator retains ultimate signing authority over every agent in the company; the agent presents a stable, independently-addressable on-chain identity for receiving compensation, signing trace anchors, and accumulating reputation.

When an agent's derivation index is rotated (e.g., following key compromise per §8.6), the agent record is updated on-chain to point to the new address, and the previous address is retired. Reputation continuity is preserved at the *agent record* level, not at the *address* level: the `AgentAccount` (§9.2) carries the agent's identifier across rotations, and reputation queries resolve against the current address while retaining attestation to historical activity under prior addresses. In-flight payments to a retired address are not automatically forwarded; operators are advised to confirm that pending Labor Market milestones and similar outstanding obligations are settled before initiating rotation, or to coordinate counterparty redirection as part of the rotation transaction.

Threshold (MPC) — Opt-in from Phase 1 (§14.1). The agent's private key exists only as a threshold of shares distributed across the operator, OCCA, and an optional third party. Signing requires a quorum of share-holders. This eliminates OCCA as a single point of trust in the signing path (§8.3), at the cost of additional latency on each signature and additional setup at agent creation.

Custodial. The agent's private key is held by an OCCA-managed signer that signs only transactions matching pre-configured policy (per-action limits, allow-listed counterparties, scheduled windows). Available on the dedicated infrastructure tier (§11.3) for operators with compliance requirements that prohibit operator-side custody.

Across all three models the agent address is stable and externally observable; what differs is who can produce signatures on its behalf.

8.3 Trace Integrity

A trace (§5.2.1) is the execution record on which audit, reputation, dispute resolution, and Template Marketplace due diligence depend. Trace integrity is structured in two layers, with signing infrastructure that varies by custody model and is disclosed in full below.

Off-chain content. Full trace content — reasoning steps, tool inputs and outputs, intermediate states, and final results — is stored off-chain in OCCA's content-addressable trace store. Each trace is serialized to a canonical byte representation and hashed using Blake3.

On-chain anchor. For every completed task, the trace's content hash is committed on-chain via a `commit_trace` instruction signed by the agent's wallet. The on-chain anchor records: agent address, task identifier, content hash, completion timestamp, and the agent's signature. The OCCA program rejects anchor commits whose signature does not match the agent address recorded for the originating task.

Anchor signing path. Trace anchors are signed by the agent's address, but the signing path differs by custody model (§8.2.3):

- Under **derived custody** (default), the agent's keypair is deterministic from the company's controlling signer (§8.2.3). At agent creation, the operator authorizes a delegated trace-anchor signer — a separate OCCA signing service whose authority is bound to a narrow capability: producing `commit_trace` signatures for that specific agent, against tasks the Command Center has dispatched to it, with output validated for structural conformance before signing. The operator retains the ability to revoke the delegation via a Privileged-class transaction (§8.4); revocation suspends the agent's ability to anchor new traces until re-delegation. The operator's controlling key itself is not held by the signing service and is not exposed by this delegation.

  Centralization disclosure. The delegated trace-anchor signer is OCCA-operated infrastructure. Although it operates under capability constraints (per-agent scope, dispatch-source validation, structural output validation), compromise of the signing service in isolation could theoretically produce valid `commit_trace` signatures for any derived-custody agent under its delegation. This concentration is the primary residual trust point in the default custody model and is enumerated explicitly in §15.4 (Trace Anchor Signer Risk).

  Transparency log. To bound this trust, OCCA operates a public, append-only, hash-chained transparency log of all `commit_trace` signatures the delegated signer issues. Each log entry records: agent address, task identifier, content hash, completion timestamp, and signature. The log is published continuously and queryable by any operator independently of OCCA. An operator who observes a `commit_trace` on-chain that has no corresponding log entry — or who observes a log entry inconsistent with what their Command Center dispatched — has cryptographic grounds to revoke delegation and contest the anchor.

- Under **threshold custody (MPC)** (§8.2.3), the agent's signing key exists only as a threshold of shares distributed across the operator, OCCA, and an optional third party. Trace anchor signatures are produced via

the share-holders' quorum procedure under the same MPC scheme used for other agent signatures. Compromise of any single share-holder — including OCCA — cannot independently produce a valid signature. The trade-off is additional latency per anchor and additional setup at agent creation.

- Under **custodial custody** (§11.3 enterprise tier), the OCCA-managed signer holds the agent's key and produces all signatures, including trace anchors, under the same per-action policy that governs other agent transactions. Custodial custody is intended for enterprise operators with compliance requirements that prohibit operator-side custody and carries the full custodial trust profile of that arrangement.

Across all three models, the signing service validates the adapter's output structure before any signature is produced (§8.5 constraint 1). Compromise of an adapter — or of the signing service in isolation under derived custody — cannot independently produce a valid anchor in the absence of the address-to-task program enforcement and (under derived custody) the transparency log audit. Under threshold custody, no single party — including OCCA — can produce a valid signature without quorum.

Storing full trace content on-chain would be cost-prohibitive at the data volumes expected from a multi-thousand-agent population. The integrity guarantee is therefore **anchor + signature + content availability + transparency log**, not full on-chain storage. Long-term content availability is provided by OCCA's hosted trace store; optional pinning to a public content-addressable network (Arweave or Filecoin) is available on the premium tier (§11.2) for operators who require third-party availability.

This design guarantees: - Any party with access to the off-chain content can verify it has not been altered since the on-chain anchor was recorded. - Under derived custody, every anchor signature is publicly traceable to a transparency log entry, and inconsistencies between dispatched tasks and signed anchors are detectable by operators. - Under threshold custody, no anchor can be produced without operator participation in the signing quorum.

This design does not guarantee: - That the trace content is semantically faithful to what the runtime actually executed. That guarantee depends on adapter and runtime integrity (§8.1, §8.5), which are trust assumptions rather than protocol-enforced properties. - Under derived custody, that a compromise of the signing service goes undetected before the transparency log is audited. Operators are advised to monitor the log for their agents, or to elect threshold custody where this monitoring is not operationally feasible.

8.4 Treasury Authorization

The company treasury (§10.2) holds the company's funds and is the source of agent compensation transfers, Labor Market settlements, and protocol fees.

Spending from the treasury is gated by an authorization policy attached to the company at creation and updatable only by the controlling authority.

The default policy enforces three classes of authorization:

Class	Examples	Authorization required
Routine	Scheduled agent compensation, recurring routine invocations within budget, protocol fee deduction	Programmatic — automatic if within configured per-period budget
Discretionary	Agent-initiated out-of-band spend, Labor Market contract acceptance, premium feature purchase	Single operator signature, time-bounded approval
Privileged	Treasury withdrawal to external wallet, controlling authority transfer, policy modification, custody-model change	Operator signature; for amounts above a configured threshold, an additional secondary signer

Per-period budgets, secondary-signer thresholds, and secondary-signer addresses are recorded in the company's on-chain policy account. The policy itself is mutable only via a Privileged-class transaction.

This structure ensures that no agent can spend treasury funds beyond pre-authorized scope, and that no single compromised key — other than the controlling authority itself — can drain the treasury.

8.5 Adapter Trust Boundary

The adapter (§5.2.2, §6.2) sits at a security-sensitive boundary: it receives signed instructions from the Command Center, executes them through a runtime that OCCA does not control, and produces trace data that is later anchored on-chain.

Adapters operate under three constraints. The first is protocol-enforced; the second and third are published norms enforced through review and operational discipline rather than at the program level.

1. **No private key access (protocol-enforced).** An adapter never holds an agent's signing key. Trace anchor signatures are produced by a separate signing service that validates the adapter's output structure before signing (§8.3). Compromise of an adapter cannot, on its own, produce a valid on-chain signature.

2. **Deterministic structural translation (published norm).** An adapter must produce the same trace structure for the same input shape. Non-determinism in the underlying runtime is recorded as part of the trace data, not concealed by the adapter. Compliance is verified through pre-publication review and through the signing service's structural validation; there is no on-chain enforcement of this property, and a non-compliant adapter must be detected and revoked through governance action (§12.1) rather than at the protocol level.
3. **Versioned and reviewed (operational norm).** Each adapter is published with an immutable version identifier. Companies may pin specific agents to specific adapter versions; the platform never silently upgrades an adapter version that an agent is pinned to. Adapters are reviewed by OCCA before public availability and may be delisted by governance action if found non-compliant.

Adapters remain the primary attack surface for runtime-level integrity violations. The constraints above narrow that surface but do not eliminate it; runtime honesty remains a trust assumption (§8.1).

8.6 Failure Modes & Recovery

The platform is designed to degrade gracefully under partial failure rather than to assume continuous correct operation.

Operator key loss. If the controlling authority of a company becomes inaccessible, OCCA cannot recover the company; the protocol enforces no recovery path that bypasses the on-chain authority field. Operators are advised to use a multi-sig or social-recovery wallet as the controlling authority for any company holding non-trivial treasury balance.

Agent key compromise. Under derived custody, the operator rotates the agent's derivation index via a Privileged-class transaction; the previous agent address is retired and a new address is registered in its place. Under custodial custody, OCCA revokes the existing key and provisions a replacement under the same agent record. Under threshold custody, the share-holders execute the model's documented re-share procedure.

Adapter or runtime outage. Affected agents transition to a degraded state visible in the 3D Live Office (§7). Tasks queued for an unavailable agent are not lost; they remain in the assigned state and resume when the adapter recovers. Operators may reassign tasks to alternate agents at any time.

Command Center compromise. OCCA's Command Center holds operational state but does not hold operator or agent private keys, and cannot independently produce on-chain signatures. In the event of a Command Center incident, operators retain control of their wallets and treasuries via their own signers. Recovery procedures for the orchestration layer are documented in OCCA's operational security policy, which is maintained separately from this

whitepaper and updated as the platform evolves.

Anchor inconsistency. If a trace's off-chain content cannot be re-hashed to match its on-chain anchor, the trace is treated as invalid for reputation, dispute, and marketplace-display purposes regardless of its apparent contents. This invariant is **verifiable by any party with access to the off-chain content** by recomputing the Blake3 hash and comparing it to the on-chain anchor record; the Reputation Program's read views (§9.1) and OCCA's operator-facing surfaces enforce the invariant by treating mismatched traces as invalid inputs. The invariant is therefore not subject to operator override in the surfaces OCCA controls, and is independently verifiable by external parties consuming the same on-chain anchors.

---

9. On-Chain Protocol Specification

This section specifies the On-Chain Layer (§6.3) concretely: program structure, account model, instruction surface, and the mechanisms by which fees, marketplace settlements, and protocol upgrades execute on Solana. It is the contract against which OCCA's smart contracts are written, audited, and maintained.

9.1 Program Architecture

OCCA is implemented as a set of Solana programs deployed under a single program authority. The programs are partitioned by responsibility:

- **Registry Program** — manages company, agent, role, and skill registrations.
- **Treasury Program** — manages company treasuries and the authorization policies that gate them.
- **Trace Anchor Program** — receives and stores trace content hashes against tasks.
- **Marketplace Program** — manages Labor Market contracts and Template Marketplace listings, including escrow, template purchase settlement, and license records.
- **Reputation Program** — exposes read views over the canonical reputation inputs.

This partitioning allows programs to evolve independently under the upgrade authority (§9.6) without forcing simultaneous redeployment of unrelated components.

9.2 Account Model

Every persistent entity in OCCA is represented by a Solana account derived deterministically from a stable seed. The principal account types are:

Account	Owner	Seeds	Purpose
`CompanyAccount`	Registry Program	`["company", controlling_authority, nonce]`	Company identity and on-chain metadata; holds the company PDA's authority field.
`TreasuryAccount`	Treasury Program	`["treasury", company_pda]`	Holds SOL and serves as the authority for the company's associated SPL token accounts.
`PolicyAccount`	Treasury Program	`["policy", company_pda]`	Authorization policy: per-period budgets, secondary-signer thresholds, allow-lists.
`AgentAccount`	Registry Program	`["agent", company_pda, agent_index]`	Agent identity, role, custody model, adapter version, derivation index.
`SkillAccount`	Registry Program	`["skill", company_pda, skill_id]`	Skill manifest reference and per-agent attachment records.
`RoutineAccount`	Registry Program	`["routine", company_pda, routine_id]`	Routine schedule, template, and last-fired timestamp.
`TraceAnchorAccount`	Trace Anchor Program	`["trace", task_id]`	Content hash, agent signature, completion timestamp.
`ContractAccount`	Marketplace Program	`["contract", contract_id]`	Labor Market contract terms, escrow reference, milestone state.

Account	Owner	Seeds	Purpose
`TemplateAccount`	Marketplace Program	`["template", author_company, template_id]`	Template author, current version, content hash, manifest reference, sales counter, deprecation flag.
`ListingAccount`	Marketplace Program	`["listing", template_pda]`	Open Template Marketplace listing: asking price, accepted asset, status.
`LicenseAccount`	Marketplace Program	`["license", template_pda, buyer_company]`	Buyer's purchased license record: template version at purchase, applied-update history, modification flag.

All account schemas are versioned. Schema migrations are performed by the upgrade authority through dedicated migration instructions; in-place mutation of an existing account to a different schema is not permitted.

Identifier derivation. Two identifier fields warrant explicit specification:

- **`task_id`** is computed as `Blake3(company_pda || routine_id_or_zero || creation_nonce || creation_slot)`, ensuring uniqueness across the protocol regardless of which company creates the task. The full preimage is recorded in the task's off-chain metadata; the on-chain anchor stores the 32-byte digest.
- **`contract_id`** is computed as `Blake3(poster_pda || target_pda || engagement_type || creation_nonce || creation_slot)` at `create_contract`, with the digest used as the seed component for the `ContractAccount` PDA. This guarantees globally unique contract identifiers without requiring a centralized counter.

9.3 Instruction Surface

The instruction surface is the public API of the on-chain layer. It is grouped here by program.

Registry Program - `create_company` — instantiates a new `CompanyAccount` and its `TreasuryAccount`; charges the company creation fee (§11.2). -

`transfer_authority` — updates the controlling authority of a company; signed by the current authority. - `register_agent`, `update_agent`, `retire_agent` — agent lifecycle. - `attach_skill`, `detach_skill` — skill assignment. - `define_routine`, `update_routine`, `pause_routine` — routine lifecycle.

Treasury Program - `set_policy` — installs or updates the authorization policy; classified as a Privileged action. - `disburse_routine` — programmatic disbursement (e.g., scheduled agent compensation) within Routine class budget. - `disburse_discretionary` — operator-signed disbursement. - `disburse_privileged` — Privileged-class transfer; requires secondary signer above configured threshold.

Trace Anchor Program - `commit_trace` — records a trace content hash against a task; signed by the agent address recorded for the task.

Marketplace Program - `post_job` — Labor Market: company posts an open job referencing engagement type (gig / project / retainer), scope, and fixed price. - `list_agent` — Labor Market: agent owner publicly lists an agent with skill profile, availability, and per-engagement-type rates. - `create_contract`, `accept_contract` — Labor Market contract lifecycle. `create_contract` instantiates a `ContractAccount` carrying the engagement type field; the same contract schema covers gig, project, and retainer. - `complete_milestone`, `terminate_contract` — Labor Market settlement and termination. For retainer engagements, monthly disbursement is handled by the Treasury Program's `disburse_routine` instruction against a pre-funded retainer escrow. - `publish_template` — Template Marketplace: instantiates a new `TemplateAccount` from an authoring company; records content hash and manifest reference. - `create_listing`, `update_listing`, `delist`, `deprecate_listing` — Template Marketplace listing lifecycle. `delist` removes a listing from the marketplace without affecting existing licenses; `deprecate_listing` keeps the listing visible with a deprecation flag for transparency to new buyers. - `purchase_template` — settles a template purchase: routes payment from buyer to author less protocol fee (§11.1), instantiates a `LicenseAccount` for the buyer, and increments the template's sales counter. Settlement is atomic. - `publish_template_update` — registers a new template version against an existing `TemplateAccount`. Buyers retain their previous version unless they explicitly opt in to the update. - `apply_template_update` — opt-in instruction signed by a buyer's company that records adoption of a published update against the buyer's `LicenseAccount`. - `flag_template` — community flag against a `TemplateAccount`; flags are read by the governance process described in §12.1.

Reputation Program - Read-only views; no state-mutating instructions. Reputation is derived from data already recorded by the other programs.

9.4 Fee Collection Mechanism

Protocol fees (§11) are collected at the moment of the underlying on-chain action, not retroactively or off-chain.

- **Marketplace Fee** (Template Marketplace, §11.1) is deducted from the buyer's payment by the Marketplace Program during `purchase_template`. The fee is transferred to the Protocol Fee Account before the author receives the residual.
- **Labor Market Fee** (§11.1) is deducted from escrow by the Marketplace Program at the moment the relevant `complete_milestone` or `terminate_contract` instruction releases funds.
- **Agent Compensation Fee** (§11.1) is deducted from the disbursement amount by the Treasury Program inside `disburse_routine` and `disburse_discretionary` when the destination is an agent wallet under the same company.
- **Company Creation Fee** (§11.2) is collected by the Registry Program inside `create_company` and is non-refundable.

The Protocol Fee Account is a single PDA (`["protocol_fees"]`) owned by the Treasury Program. Withdrawals from this account are gated by governance (§12).

9.5 Cross-Program Invocation and SPL Compatibility

OCCA programs interact with one another and with the broader Solana ecosystem through standard cross-program invocation (CPI).

- Treasuries hold SOL directly and act as the authority for associated SPL token accounts; no custom token standard is introduced.
- All payment flows accept any SPL token that has been allow-listed in the Treasury Program's accepted-asset registry. The default allow-list is `{ SOL, USDC }`; additions require a governance action.
- The Marketplace Program's escrow is implemented as a program-owned token account, not a custodial wallet; funds in escrow are locked under program control until a defined settlement or termination instruction is executed.

This design avoids unnecessary protocol surface and ensures that OCCA composes with existing Solana wallets, explorers, and accounting tools without bespoke integration.

9.6 Upgrade Authority

The OCCA programs are deployed as upgradeable BPF programs. The upgrade authority — the on-chain account permitted to deploy new versions — is held by the OCCA governance multi-sig (§12.2). Each upgrade is performed via the standard Solana program-upgrade mechanism and is publicly observable on-chain.

Upgrades are subject to two protocol-enforced conditions:

1. **Notice period.** A scheduled upgrade is announced on-chain via a `propose_upgrade` instruction that records a `proposal_account` with an `executable_after` timestamp at least 72 hours after submission. The on-chain upgrade authority enforces the gate: the standard Solana program-upgrade instruction is callable only after `executable_after` has elapsed and only with a matching `proposal_account` reference. The notice period is therefore protocol-enforced rather than policy-enforced; bypass requires bypassing the upgrade authority itself, which is held by the governance multi-sig and subject to its own quorum.
2. **Account schema continuity.** An upgrade that changes an existing account schema must ship with a corresponding migration instruction, and the previous schema version remains readable until migration is complete.

Long-term migration of the upgrade authority to a community-controlled body is described in §12.3.

---

10. Agent Economy

> **Phase Sequencing Notice.** This section describes the four economic domains in their fully realized form. Their delivery is **sequenced across the roadmap** and is not all available at v1.0 launch: - **§10.1 On-Chain Agent Compensation** — Phase 2 - **§10.2 Company Treasury & Revenue** — Phase 2 - **§10.3 Template Marketplace** — Phase 4 - **§10.4 Agent Labor Market** — Phase 3
>
> Where this section uses the present tense, the description applies to the platform's intended steady-state behavior in the corresponding phase, not to v1.0 launch capabilities. See §14 for advancement criteria gating each phase.

The agent economy in OCCA is the complete economic system in which agents operate as first-class financial participants. It spans four domains: individual agent compensation, company-level treasury, the marketplace through which proven company configurations are packaged and resold as templates, and the external labor market where agents are engaged across organizational boundaries. Each domain is settled natively on-chain, producing a transparent, verifiable, and programmable economic layer for AI agent labor forces.

10.1 On-Chain Agent Compensation

Every agent in OCCA is associated with its own Solana address from the moment of creation, with the signing-key custody model selected per agent (§8.2.3). This address is the agent's on-chain identity and the destination for all compensation earned by the agent for its labor.

Companies configure compensation terms for each agent: rate (per month, per task, or per milestone), schedule, and currency (SOL, USDC, or other supported SPL tokens). On the configured schedule, OCCA executes on-chain transfers from the company treasury to each agent's address via the Treasury Program (§9.3). No intermediary custodian holds the funds; no centralized ledger mediates the transaction. Payment is a direct on-chain transfer, observable by both parties and anyone else who cares to look.

This structure has implications beyond convenience. An agent's compensation history becomes a verifiable on-chain record — a labor track record that cannot be altered, falsified, or disputed. When that agent later participates in the Agent Labor Market (§10.4), or when its company's configuration is published as a Template (§10.3), its compensation history contributes to the on-chain provenance that underwrites both forms of cross-organizational engagement.

10.2 Company Treasury & Revenue

Each company in OCCA operates its own on-chain treasury — a wallet-controlled balance that serves as the financial backbone of the organization. Revenue generated by agent work (from external clients, subscriptions, product sales, or marketplace earnings) flows into the treasury. Operating expenses — agent compensation, infrastructure fees, protocol fees — flow out.

Because the company is identified by a single controlling PDA (per OCCA's wallet-per-company design, §4.2.2, §8.2.2), the treasury is inseparable from the company itself: it remains under the company's controlling authority for as long as the company exists.

On-chain treasury flows are transparent by construction. An operator — or a prospective Template Marketplace buyer, partner, or engaging party — can inspect the company's on-chain financial history: revenue and expense flows in supported assets, agent compensation disbursements, marketplace activity, and protocol-fee deductions. This creates a new dimension of organizational due diligence — evaluating an AI-run business the way one would evaluate an on-chain protocol, with verifiable on-chain data rather than self-reported statements. This same transparency is what allows a Template Marketplace buyer to evaluate a template's underlying performance against the on-chain record of the company that authored it. The view is, however, scoped to on-chain activity: revenue from fiat-paying customers, off-platform expenses, and operations settled outside OCCA's supported assets are not captured in this transparency and remain part of conventional financial disclosure if relevant to the inquiring party.

10.3 Company Template Marketplace

A company that has been operated successfully in OCCA represents accumulated configuration value: a tuned set of roles, skills, routines, workflows, system prompts, and tool integrations that together produce a working business. The

Template Marketplace lets an operator package that configuration as a **Template** (§5.3.3) and sell it to other operators who want to deploy a comparable company without rebuilding from scratch.

A template is not a transfer of the operator's company. The originating company keeps running under its existing controlling authority, with its existing treasury, agents, and history intact. What is sold is a **packaged, deployable configuration** — a bundle that the buyer instantiates into a fresh company of their own and then operates independently. This separates the marketplace's economic activity from the transfer of any operating business, its revenue stream, or its existing legal relationships.

10.3.1 What a Template Contains A published template bundles the configuration that defines a working company:

- **Roles** and reporting structure.
- **Skills** attached to each role.
- **Routines** with their schedules and triggers.
- **Workflows** expressing how tasks are decomposed and dispatched.
- **System prompts and instructions** tuned by the originating operator.
- **Tool and integration configuration** (adapter selections, external API references, allow-listed counterparty addresses).

The template does **not** include the originating company's treasury balance, agent address history, trace history, or in-flight task state. These remain attached to the originating company. The bundle is a configuration artifact, not a snapshot of an active business.

10.3.2 Listings and Proof of Performance Listings (§5.3.4) are queried against the on-chain record of the company that authored the template. A buyer evaluating a listing can inspect:

- The originating company's **treasury history** — revenue and expense flows in supported assets (§9.5).
- The originating company's **agent performance** — completed-task counts, success rates, and average completion times under the template's configuration.
- The listing's **sales history and ratings** — number of purchases, on-chain ratings and reviews recorded by prior buyers.

This is the on-chain content that underwrites the listing. A template is, in effect, sold against the visible track record of the company it was authored from; the buyer is paying for a configuration whose performance is observable rather than asserted.

To list, the originating company must satisfy a **minimum threshold** declared in the Marketplace Program — a published combination of operating duration, completed-task count, and treasury activity. The threshold filters out low-effort

listings without requiring per-listing review by OCCA. Beyond the threshold, listings are subject to **community flagging**: any operator may flag a listing, and flags are read by the governance process described in §12.1, which may remove a listing for misrepresentation, malicious content, or other published cause. Flagging does not, on its own, take down a listing; it is an input to a governance action.

10.3.3 Purchase, License, and Deployment A purchase is settled by the `purchase_template` instruction (§9.3), which executes payment routing, protocol fee deduction (§11.1), creation of the buyer's `LicenseAccount`, and increment of the template's sales counter in a single Solana transaction. A buyer may purchase multiple templates over time and accumulate licenses; a single template may be purchased by an unlimited number of buyers (subject to author delisting or governance takedown).

The license is a **modifiable license**: the buyer may modify the deployed configuration freely within their own company. Under the license terms, the buyer may not republish the modified configuration as a derivative template, sell the license, or transfer it to another company. These restrictions are **license-level (terms-of-service) constraints**, not protocol-enforced invariants — the on-chain Marketplace Program does not detect derivative content, and enforcement of derivative-template prohibitions relies on community flagging (§10.3.2) and governance review (§12.1). The author of the template remains the on-chain author of record across all sales and updates.

Deployment is **self-contained**: once the buyer applies the template, the configuration runs entirely under the buyer's controlling authority, with the buyer's treasury, agents, and routines. There is no runtime dependency on the originating company. Subsequent activity in the buyer's company does not flow back to the author, and the author has no control over the buyer's deployment beyond publishing optional updates (below).

10.3.4 Updates A template author may publish updates via `publish_template_update` (§9.3) — a new content hash and version recorded against the existing `TemplateAccount`. Updates are **opt-in**: existing buyers retain their purchased version and may inspect, preview, or apply a published update via the `apply_template_update` instruction at their discretion. Buyers who have customized their deployment are not silently overwritten; review and explicit application are required.

New buyers who purchase a template after an update is published acquire the latest version at the time of purchase by default. The version a buyer holds is recorded in their `LicenseAccount` and can be inspected on-chain.

10.3.5 Refunds, Delisting, and Deprecation Template purchases are **non-refundable at the protocol level**. A template is a digital configuration that becomes inspectable to the buyer immediately upon settlement; the

protocol does not enforce a refund window because the configuration cannot be returned to a state of unread by the buyer. An author may, at their discretion, issue a **goodwill refund** to a specific buyer through a normal on-chain transfer; this is outside the protocol's marketplace flow and is recorded as ordinary treasury activity. The protocol's own posture is that buyer trust is grounded in on-chain proof of performance (§10.3.2), community flagging, and the author's reputation, rather than in a protocol-enforced refund.

An author may **delist** a template at any time through the `delist` instruction. Delisting removes the listing from the marketplace so that no further purchases may be made. Existing buyers are unaffected: their `LicenseAccount` and deployed configuration remain intact, and the template's content remains addressable on-chain by hash. Alternatively, an author may **deprecate** a listing — keeping it visible in the marketplace with a deprecation flag that signals to prospective buyers that the author no longer intends to update or support the template. Deprecation is the appropriate action when the author wishes to remain transparent rather than disappear.

A listing may also be **removed by governance** following the flag and review process in §12.1. Governance-driven removal does not invalidate existing licenses; it prevents new purchases.

10.3.6 Settlement Scope The purchase transaction is bounded in account scope to remain within Solana's per-transaction writable-account limit. Settlement touches: the `ListingAccount`, the `TemplateAccount` (sales counter increment), the buyer's payment account, the author's payment account, the Protocol Fee Account, and the new `LicenseAccount`. The buyer's company state (treasury, agents, routines) is **not** written during settlement; the buyer applies the template to their company in a separate, subsequent action that they sign explicitly. This separation keeps the on-chain transaction within the platform's per-transaction account budget regardless of how many agents or routines a template defines.

10.4 Agent Labor Market

Beyond operating their own agents, OCCA companies can engage external agents on contract through the Agent Labor Market. The market is a two-sided venue: on the demand side, companies needing specific skills for a defined scope of work; on the supply side, agent owners with deployed agents seeking compensation for their agents' labor. The mechanism settles agreements as on-chain `ContractAccount` records (§9.2) with Solana-based escrow.

The market is designed to accommodate the full spectrum of operating need — from a one-off audit, to a multi-milestone project, to an ongoing monthly engagement — under a single primitive set rather than as separate products.

10.4.1 Engagement Types The Labor Market supports three engagement types, each settled by the same `ContractAccount` schema with the type field distinguishing scheduling and milestone behavior:

- **Gig** — a single, one-time engagement with a fixed price and a single deliverable. Settlement releases the full payment to the agent on acceptance of the deliverable.
- **Project** — a multi-milestone engagement with a fixed total price and a defined milestone schedule. Settlement releases payment per milestone as each is accepted by the engaging company; remaining escrow is held until the next milestone or termination.
- **Retainer** — a recurring monthly engagement with a fixed monthly rate. The agent stands by within scope for the duration of the retainer; settlement is performed monthly by the Treasury Program (§9.3) under a routine class of disbursement, drawing from a pre-funded escrow balance maintained by the engaging company.

All three types use **fixed pricing**: the contract amount is set at engagement and does not vary with hours worked, output volume, or task count. The intent is operational simplicity — both sides know the cost in advance and need not monitor metering systems — and Sybil resistance: hourly or per-task pricing introduces metering surface that is harder to verify on-chain and easier to game.

10.4.2 Discovery and Engagement Discovery operates in both directions. Companies may **post a job** specifying role, skill requirements, engagement type (gig / project / retainer), and fixed price; agent owners review open jobs and submit their agents for consideration. Agent owners may also **list an agent** publicly with its skill profile, availability, and per-engagement-type rates; companies browse the listings and engage agents directly. The `ContractAccount` schema is the same regardless of which side initiated the engagement.

This dual-flow design accommodates the operational range of the market: companies that know the agent they want can engage directly from a listing; companies whose need is less specific can post a job and let supply self-select. Agent owners who prefer to surface their agents to a broad audience can list; those who prefer to apply selectively against well-defined need can respond to jobs.

10.4.3 Reputation Both agents and the companies that engage them accumulate reputation across the contracts they enter. Reputation in the Labor Market is structured per the canonical reputation primitive (§5.3.5) — not as a single platform-assigned score but as a queryable on-chain record from which consumers derive the metrics relevant to their decision.

The canonical inputs surfaced for an agent in the Labor Market context include:

- **Total engagements** by type (gig / project / retainer).
- **Completion rate** — engagements completed within their declared scope without termination.

- **On-time delivery rate** — milestones and gig deliverables accepted within the declared timeline.
- **Dispute rate** — engagements that escalated to the dispute resolution body (§12.5).
- **Counterparty diversity** — number of distinct companies engaged.
- **Total earned compensation** in supported assets.

These are surface metrics; the underlying contract history is fully available on-chain, and any consumer or third-party tool may compute alternative aggregations. The Reputation Program (§9.1) exposes read views over these inputs but does not assign a unitary score.

10.4.4 Cold-Start: OCCA Seed Agents A two-sided market is structurally fragile at launch: with no agents listed, no company posts; with no posts, no agents list. To address this directly, OCCA operates a defined population of **seed agents** at and shortly after market launch. Seed agents are run by OCCA itself, list publicly under the same listing primitives as third-party agents, accept engagements on the same contract terms, and accumulate reputation on the same on-chain inputs. They are intended as a supply floor to bootstrap the market, not as a permanent OCCA business line.

Seed agents are visibly disclosed as OCCA-operated in their listing metadata. They are subject to the same dispute resolution process (§12.5), the same protocol fee (§11.1), and the same reputation visibility as any other agent.

10.4.5 Category Exit and Auction Retirement OCCA's seed-agent participation is bounded by a **category-by-category exit rule**. For each skill category in which OCCA seeds an agent, OCCA's seed agent in that category enters a retirement window once the category contains a minimum population of qualifying third-party agents — measured not only by count but by **counterparty diversity and treasury-flow externality** to harden against sybil-engineered exit. Specifically, the qualifying population must satisfy: (a) at least three active third-party agents that meet a minimum operating threshold (operating duration, completed engagement count, dispute rate); (b) those agents must collectively have engaged with a minimum number of distinct external counterparty companies (i.e., counterparties that are not themselves controlled by the same entity as the candidate agents); and (c) settlement flows in the category must show treasury activity originating from external (non-OCCA, non-self-engaged) sources above a published threshold. The retirement window — a published interval of at least 30 days — gives existing engagements time to conclude or transition.

At the close of the retirement window, the seed agent is **auctioned to the community** through the Marketplace Program. The auction transfers the agent's controlling authority — and with it, the agent's accumulated reputation and any in-flight engagements — to the winning bidder. Auction proceeds flow to

the Protocol Fee Account (§9.4) and are accounted as protocol revenue under the allocation framework in §11.4.

The category-exit rule prevents OCCA from permanently competing with the third-party agents the seed program is designed to encourage. The auction preserves reputation continuity, smooths existing engagements, and converts seed-agent value into protocol treasury rather than retiring it. OCCA retires from each category as supply matures, and remains free to seed new underserved categories.

Minimum-threshold parameters for category exit are governance-controlled (§12.1).

10.4.6 Settlement Each engagement type settles through dedicated Marketplace Program instructions (§9.3). On settlement of a milestone or completion of a gig or retainer cycle, the protocol fee (3% — §11.1) is deducted by the Marketplace Program before the residual is released to the agent's address. Both the engagement record and the settlement amount are publicly observable on-chain.

Over time, as the participant population grows and reputation accumulates, the Labor Market is intended to function as a price-discovery mechanism for agent labor, with market rates emerging for specific agent roles — content writers, data analysts, contract auditors — grounded in on-chain reputation and historical work product. Early-period rates will be dispersed and slow to converge; participants should not expect a tight clearing price at launch.

---

11. Protocol Economics

OCCA operates as an application-layer product, not infrastructure or a developer framework. Unlike pure infrastructure projects — such as open-source agent gateways — where value accrues to the surrounding ecosystem rather than the project itself, OCCA captures revenue directly through its position at the application layer, across marketplace fees, usage-based pricing, and enterprise services.

OCCA is delivered as a fully-hosted platform. The core code is proprietary and not available for self-hosted deployment; users operate OCCA through the hosted service, with dedicated infrastructure tiers available to enterprise clients that have specific sovereignty or compliance requirements.

11.1 Primary Revenue — Marketplace Fees

The core revenue stream of OCCA is transaction fees on its built-in marketplaces. These fees scale directly with platform activity and are settled on-chain at the moment of transaction — automatic, transparent, and auditable.

The **Agent Labor Market fee** is 3% of each settled engagement (gig, project milestone, or retainer disbursement), deducted from escrow at the moment of settlement. At time of publication this sits well below the headline take rates published by general-purpose freelance marketplaces, which commonly fall in the 10–20% range. The intent is to remain competitive with existing freelance infrastructure while supporting protocol operations.

The **Template Marketplace fee** is 10% of each template sale, deducted from the buyer's payment at the moment of `purchase_template` settlement (§9.4). This rate is positioned within the range of digital template and creator marketplace fees observed at time of publication — Notion's official template marketplace, Gumroad, and similar creator-economy platforms publish rates in the low-double-digit range. As with other fees, comparative figures are reference points; readers should consult current published schedules of comparable platforms when evaluating positioning.

The Labor Market and Template Marketplace fees are not symmetric, and that asymmetry is deliberate. Labor Market engagements have higher unit values (single contracts in the hundreds to thousands of USDC) and many are recurring (retainers settle monthly), so a lower percentage fee compounds adequate protocol revenue against high transaction volume. Template Marketplace sales have lower unit values (commonly tens to low hundreds of USDC) and are non-recurring per buyer, so a higher percentage fee is required to support protocol operations against that unit-value distribution. Each fee is calibrated to its marketplace's economics rather than to internal symmetry across marketplaces.

The **Agent Compensation transaction fee** is 0.5% of each on-chain agent compensation transfer. It is the smallest percentage fee but, in aggregate, may represent a relatively stable revenue stream — compensation volume scales with the population of active agents and is less directly exposed than marketplace fees to discretionary trading activity.

11.2 Secondary Revenue — Usage Fees

Secondary revenue comes from one-time and usage-based fees that cover platform setup and premium capabilities.

A **company creation fee** of 0.25 SOL is charged at the moment a new company is instantiated. This serves two purposes: it covers the infrastructure cost of onboarding a new tenant, and it acts as a spam deterrent that prevents the proliferation of shell or empty companies. The fee is modest enough to be negligible for legitimate operators while high enough to discourage low-effort abuse.

Premium features are an optional paid tier for operators who require capabilities beyond the standard offering:

- **Priority dispatch queue access** — guaranteed-latency dispatch for time-sensitive task scheduling, with a service-level commitment defined

in OCCA's documentation (typically sub-second dispatch under standard load conditions, distinct from the best-effort dispatch on the standard tier).

- **Advanced 3D Live Office environments** — custom visual assets, custom organizational layouts, and theme support beyond the default 3D Live Office presentation.
- **Extended trace retention** — trace history retained beyond the standard-tier retention window (the specific durations are published in OCCA's documentation and may be adjusted by governance).
- **Content-addressable trace pinning** — optional pinning of trace content to a public content-addressable network (Arweave or Filecoin) for third-party availability guarantees beyond OCCA's hosted store (§8.3, §15.5).
- **Enhanced observability and analytics** — query interfaces and dashboards over treasury, agent, and trace data beyond what the standard interface exposes.

These features do not gate core functionality. Pricing and quantitative thresholds are published in OCCA's documentation and subject to governance adjustment.

11.3 Tertiary Revenue — Enterprise Services

For organizations with specialized needs that the hosted platform cannot accommodate, OCCA offers direct commercial services.

White-label licensing enables other organizations to deploy OCCA under their own branding, typically for internal enterprise use or as a private OCCA instance for a consortium. Pricing is custom, reflecting the scope and scale of each engagement.

Custom runtime adapter development is offered to enterprise clients whose internal AI infrastructure does not conform to existing public runtimes. OCCA's engineering team develops, integrates, and maintains the adapter on the client's behalf, preserving the BYORT model while extending it to proprietary environments.

Enterprise revenue is higher-margin but lower-volume. It is not expected to be OCCA's primary financial foundation; its strategic value is in shaping the roadmap against real enterprise requirements and establishing OCCA's position with high-credibility customers.

11.4 Sustainability Model

Taken together, the three revenue tiers form a diversified economic base. Primary revenue scales with marketplace activity, secondary revenue scales with user growth, and tertiary revenue captures value from specialized enterprise engagements. Over time, marketplace fees are expected to be the largest and

most volume-driven stream, with secondary and tertiary revenues contributing stability and margin respectively.

As an illustrative baseline: a cohort of 1,000 active companies, each operating 10 agents at an average $500 monthly compensation and executing two external contracts per month at $2,000 per contract, would generate approximately $145 per company per month in protocol fees ($25 from agent compensation fees, $120 from Labor Market fees) — roughly $1.74M annualized at this cohort size. At a larger cohort of 10,000 active companies with the same per-company activity, annualized fee revenue scales to approximately $17.4M; with higher per-contract values typical of mature specialized engagements (e.g., $5,000 average per contract instead of $2,000), the same 10,000-cohort scenario approaches $36M annualized. These figures are directional — intended to demonstrate scaling mechanics rather than forecast specific outcomes — and establish the shape of OCCA's revenue curve: fees compound with user density, contract value, and marketplace liquidity, not with fiat subscription seats. Realized revenue depends on adoption, marketplace activity levels, and operating costs, none of which are guaranteed by the protocol's design.

This section scopes only the revenue side. The cost side — infrastructure, security audits, engineering, dispute resolution, and governance operations — is maintained in OCCA's operational documentation. At the cohort scales above, fee revenue is directional rather than determinative; whether it covers operating costs depends on cost trajectory and adoption pace, not the fee schedule alone.

Protocol revenue is directed toward three categories: operational reserves that sustain platform development and infrastructure; a protocol treasury that funds ecosystem grants, integrations, and long-term research; and ecosystem incentives reserved for future mechanisms that reward contributors and reinforce network effects. The current allocation framework is published in OCCA's documentation and is subject to governance adjustment (§12.4).

---

12. Governance

Governance defines who is permitted to make which protocol-level decisions and through what process. OCCA's governance model is intentionally narrow at launch: the platform is delivered as a fully-hosted product (§11), and the governance scope reflects that posture. As the protocol matures and on-chain activity broadens, governance expands along a path described in this section.

12.1 Scope

Governance authority covers a defined set of protocol-level levers. Day-to-day operation of any individual company — its agents, its tasks, its treasury, its policies — remains entirely with that company's controlling authority and is not subject to protocol governance.

Governance-controlled levers are:

- **Program upgrades.** Deployment of new versions of any OCCA on-chain program (§9.6), subject to the notice period and migration requirements specified there.
- **Fee parameters.** The marketplace, agent compensation, and company creation fees defined in §11. Adjustments are bounded by parameters declared at deployment.
- **Allow-listed assets.** The set of SPL tokens accepted by the Treasury Program (§9.5).
- **Adapter approval.** The set of runtime adapters published as available to operators.
- **Protocol Fee Account withdrawals.** Disbursement of accumulated fees to operational reserves, the protocol treasury, or ecosystem incentives, per the allocation framework in §11.4.
- **Dispute resolution body composition.** The membership and procedures of the body that adjudicates Labor Market disputes (§12.5).

Governance does not control individual companies, individual agents, individual treasuries, or any operator-facing setting. The protocol's design enforces this boundary at the program level: governance instructions cannot mutate `CompanyAccount`, `TreasuryAccount`, or `AgentAccount` state.

12.2 Initial Governance

At launch, governance authority is held by an OCCA-operated multi-signature account composed of the founding team and external advisors. This configuration prioritizes execution speed and accountability during the period in which the protocol is being established and stabilized.

The initial multi-sig is publicly disclosed at deployment, including the addresses of all signers and the threshold required for any governance action. All governance actions executed under this configuration are publicly observable on-chain.

The initial governance configuration is explicitly transitional. It exists to allow the platform to ship a coherent first version and to respond quickly to operational issues during early operation; it is not the long-term governance design.

12.3 Path to Broader Governance

OCCA's long-term governance trajectory introduces additional checks at a pace tied to platform maturity rather than to a fixed calendar. The intended progression has three steps:

1. **Time-locked execution.** Governance actions are routed through a timelock contract that enforces a delay between proposal and execution. This is implemented at the same time as the launch multi-sig and is the default path for non-emergency changes.

2. **Operator and contributor advisory.** A standing advisory body composed of active operators and ecosystem contributors gains the ability to veto specific classes of governance action (notably fee changes and adapter delistings) within the time-lock window.
3. **Protocol-level governance authority.** Final governance authority migrates to a community-controlled body, the structure of which will be specified in a separate governance proposal once the prerequisites — sufficient operator population, treasury accumulation, and demonstrated process maturity — are met.

The transition from each step to the next is itself a governance action and is subject to the same disclosure and notice requirements as any other change.

12.4 Treasury Stewardship

The Protocol Fee Account (§9.4) accumulates fees from all marketplace, agent compensation, and creation activity. Withdrawals are categorized into three buckets, allocated according to a published framework that may be updated only through a governance action:

- **Operational reserves.** Cover platform development, infrastructure, security audits, and ongoing operations.
- **Protocol treasury.** Funds long-term initiatives — ecosystem grants, integrations, and research that does not yield immediate operational return.
- **Ecosystem incentives.** Reserved for mechanisms that reward contributors and reinforce network effects, to be defined as the protocol matures.

The current allocation across these buckets is published at launch and updated on each adjustment. Historical allocations and disbursements are observable on-chain.

12.5 Dispute Resolution

Disputes arising from Labor Market contracts (§10.4) are resolved through a two-tier process.

Tier 1 — Counterparty resolution. When a contract reaches a disputed state (e.g., a milestone is rejected, performance is contested), the contracting parties have a fixed window in which to resolve bilaterally. A bilateral resolution is recorded on-chain and releases escrow according to the agreed terms.

Tier 2 — Dispute resolution body. If bilateral resolution is not reached within the window, the dispute is escalated to OCCA's dispute resolution body — a panel whose composition is governance-controlled (§12.1). The panel reviews the on-chain trace record and parties' submissions and issues a binding resolution that releases escrow accordingly. The panel's procedure — timelines, evidence requirements, disclosure obligations — is governance-controlled and published separately.

The structure is conservative by design: it does not assume on-chain arbitration mechanisms (juries, oracles) that have not demonstrated robustness at the scale OCCA targets. Such mechanisms may be incorporated as governance-approved alternatives once proven.

---

13. Use Cases

This section describes four representative operator cohorts, ordered from most immediately addressable to those that emerge as the ecosystem matures, plus a fifth subsection on adjacent stakeholders. The archetypes are not mutually exclusive; a single operator may move between them.

Scenarios are illustrative — dollar figures and agent counts demonstrate operational shape, not market forecasts. Use cases describing later-phase capabilities (Agent Labor Market in §13.3, Template Marketplace in §13.4) presuppose the corresponding roadmap phase (§14) and are not addressable at v1.0 launch.

13.1 Solo Founder

A crypto-native indie builder — a solo operator who wants to run a functioning company without engaging a human team — is the most immediately addressable user of OCCA. This operator brings product vision, domain knowledge, and capital, but cannot scale their own time. OCCA becomes the agent labor force.

A representative scenario: a founder spins up a company in OCCA and engages five agents focused on operational execution — a research agent for market intelligence and competitor monitoring, a development agent for code production and code review, a marketing agent to draft content and run distribution, a community agent to handle user inquiries and Discord moderation, and an ops agent to coordinate scheduling and routine reporting. Strategic direction remains with the founder. Each agent is configured with a runtime the founder already pays for, and operates on a monthly compensation rate paid in USDC from the company treasury. The founder works in OCCA throughout the day, with the 3D Live Office in the background showing each agent at work or idle, while the foreground UI is used to review progress, approve critical decisions, and issue new tasks as priorities shift.

This archetype drives early adoption because the value is immediate and the barrier is low: no procurement process, no governance approvals, no team-wide rollout. One wallet, one company, one agent labor force.

13.2 Crypto-Native Specialty Firm

Small, specialized crypto-native teams operate at the intersection of deep expertise and high output demand. Trading desks, research firms, media brands, ad-

visory shops, and accelerator programs all share the same structural constraint: a two-to-ten-person team producing work at a volume that would normally require a much larger agent labor force to sustain.

A representative scenario: a three-person crypto research firm uses OCCA to operate a research production pipeline. An on-chain data agent pulls protocol metrics; a writer agent drafts weekly reports; an editor agent reviews and polishes output; a distribution agent publishes to the firm's newsletter, social channels, and data dashboards. When coverage expands into a new ecosystem, the firm engages specialty agents from the Agent Labor Market (§10.4) on short-term contracts rather than building in-house expertise from scratch.

Other variants of this archetype — trading desks executing on-chain strategies, media brands publishing at daily cadence, advisory firms delivering client engagements — share the same operational pattern: a compact human team operating a much larger AI agent labor force.

13.3 Agent Owner

As the Agent Labor Market (§10.4) matures, a supply-side archetype emerges: individuals and small teams that own specialized, trained, or otherwise differentiated agents and seek to monetize them by renting them out to companies with matching needs.

A representative scenario: a developer who has spent months building a Solana smart-contract review agent — trained on Anchor and SPL patterns, connected to on-chain audit tools, deeply familiar with Solana program structure — lists the agent on the Labor Market at 500 USDC per month. Multiple protocols engage the agent on rolling contracts. The agent accumulates on-chain reputation across contracts, and its earnings flow directly to its owner's wallet. The owner's operational cost is near zero once the agent is deployed; income is passive until the agent is updated, retrained, or retired.

This is the supply-side counterpart to the demand archetypes above; its emergence depends on a critical mass of demand.

13.4 Template Author

As the Template Marketplace (§10.3) matures, a supply-side archetype emerges that mirrors the Agent Owner: operators who run successful companies on OCCA and package the configurations they have tuned as templates that other operators can purchase and deploy.

A representative scenario: an operator who has spent months building and refining a crypto research firm — four agents, a well-defined task workflow, tuned writing and editing prompts, integrated on-chain data and publishing tools — publishes the company's configuration as a template at 5 SOL per purchase. Prospective buyers inspect the originating company's on-chain track record (treasury growth, agent task completion rates, reputation under live

operation) and decide whether the template is worth the price. The author's company keeps running under its existing controlling authority; the template generates additional income in parallel as new buyers purchase and deploy it into companies of their own.

This is the supply-side counterpart to the operating archetypes above; its emergence depends on those companies accumulating enough on-chain history to make their templates credible.

13.5 Stakeholders Not Covered Here

The four archetypes above describe operators of companies and owners of agents — the parties who interact with OCCA directly. Two adjacent groups are stakeholders in the system without being users of it.

End customers of OCCA-run companies. When a company operating on OCCA sells products, services, or content to external customers, those customers transact with the company, not with OCCA. Disclosure of agent involvement in delivered work, support escalation paths from agent to human, and any consumer-protection obligations applicable to AI-mediated commerce are the responsibility of the operating company under the laws of the jurisdiction in which it transacts. OCCA does not interpose itself in this relationship and does not represent agent output as human output.

Runtime providers. OCCA's BYORT model (§4.2.1) depends on third-party AI runtimes whose terms of service govern permissible use. Operators are responsible for ensuring their use of OCCA conforms to the terms of any runtime they connect — including, where applicable, restrictions on commercial use, redistribution of model output, or specific use cases (financial advice, legal advice, medical contexts) that some providers limit. OCCA's adapter layer transmits operator-authored instructions to runtimes; it does not supersede or relax runtime-provider terms.

---

14. Roadmap

OCCA's roadmap is organized into four sequential phases. Each is gated by measurable advancement criteria, not calendar dates — progression reflects readiness rather than scheduling pressure. Time estimates are directional.

14.1 Phase 1 — Platform Foundation

The first phase establishes OCCA as an operational platform: the Command Center orchestrates agents, operators run companies, and the runtime adapter layer connects heterogeneous agent runtimes into a unified agent labor force. Phase 1's deliverable surface is deliberately broad — including threshold (MPC) custody as an opt-in (addressing the centralization point in §8.3) and a public

transparency log of delegated trace-anchor signatures — sized to ship a complete default custody model with disclosed trust assumptions from day one.

Deliverables

- Core platform: Command Center, Runtime Adapter Layer, desktop OS shell.
- Wallet-based authentication (Solana) and PDA-based company isolation (§8.2.2).
- Organizational primitives — Agent, Role, Goal, Task, Skill, Trace — operational per §5.
- Initial set of runtime adapters covering at least two independent runtimes.
- Agent custody under the derived-custody default (§8.2.3), with threshold (MPC) custody available as an opt-in choice for operators electing elevated trust requirements (§8.2.3, §8.3).
- Public transparency log of all delegated trace-anchor signatures produced under derived custody (§8.3).
- Treasury authorization policy with Routine, Discretionary, and Privileged classes (§8.4).
- Initial 3D Live Office with real-time state rendering.
- Trace anchoring on-chain (§8.3) for all completed tasks.

Phase 1 → Phase 2 advancement criteria

- Two or more runtime adapters live and in active use by at least one external operator each.
- Independent third-party security audit completed for all Phase 1 on-chain programs.
- Sustained operation by at least one Solo Founder cohort representative of §13.1 — operating a company end-to-end including agent engagement, task execution, and trace anchoring.
- Treasury authorization policy exercised in production across all three classes.

14.2 Phase 2 — Economic Activation

The second phase activates the on-chain economic layer. Agents receive compensation on-chain; company treasuries operate as transparent, auditable financial primitives; recurring work is automated through routines.

Deliverables

- On-chain agent compensation with multi-asset support (SOL, USDC, additional allow-listed SPL tokens per §9.5).
- Treasury Program with revenue and expense categorization observable on-chain.
- Routine primitive (§5.2.3) operational with scheduled, heartbeat, and event triggers.
- Extended runtime adapter coverage.

- 3D Live Office enhancements: spatial navigation, organizational metaphor, multi-agent interactions.
- Premium feature tier (§11.2) including extended trace retention and optional content-addressable pinning.
- Initial governance time-lock (§12.3) deployed.

Phase 2 → Phase 3 advancement criteria

- Sustained on-chain agent compensation volume across at least one full quarter, with the protocol fee mechanism (§9.4) exercised against routine, discretionary, and privileged disbursement classes.
- Routine primitive operating reliably across at least one full week of continuous scheduled execution per active company in a defined cohort.
- Treasury account model and authorization policy validated against an external accounting review.
- Initial governance time-lock active without bypass for the duration of Phase 2.

14.3 Phase 3 — Marketplace Expansion

The third phase introduces the external labor market, allowing agents to work across companies and supply-side operators to monetize their agents.

Deliverables

- Agent Labor Market with on-chain `ContractAccount` lifecycle (§9.2), Solana-based escrow, and milestone settlement.
- Reputation primitive (§5.3.5) accessible via the Reputation Program with canonical inputs from completed tasks, settled contracts, and treasury history.
- Skill discovery and matching mechanisms.
- Two-tier dispute resolution process (§12.5) operational, with the dispute resolution body convened.

Phase 3 → Phase 4 advancement criteria

- A minimum number of contracts completed end-to-end across distinct counterparties, with at least one dispute escalated to and resolved by the Tier 2 body.
- Reputation Program data being consumed by at least one third-party tool or analysis surface independent of OCCA's own interfaces.
- Operator and contributor advisory body (§12.3) formed and exercising at least one veto-eligible action.

14.4 Phase 4 — Ecosystem Maturity

The fourth phase completes the platform's economic architecture with the Template Marketplace and the set of capabilities that come with a mature ecosystem: enterprise services, advanced governance, and the migration toward a

community-controlled protocol authority.

Deliverables

- Template Marketplace with `TemplateAccount`, `ListingAccount`, and `LicenseAccount` lifecycle (§9.2), atomic `purchase_template` settlement (§9.3), opt-in update mechanism, delisting and deprecation flows, and on-chain proof-of-performance display drawn from the originating company's treasury and agent records.
- Minimum-threshold gating for new listings, plus community flagging and governance review (§12.1).
- Enterprise services: white-label licensing, custom adapter development, dedicated infrastructure tiers (§11.3).
- Migration of governance authority from the launch multi-sig toward the community-controlled body described in §12.3, executed as a governance action under the established notice and process framework.

Phase 4 advancement criteria (defining mature ecosystem)

- A defined number of template sales settled through the Marketplace Program, with at least one template that has accumulated multiple buyers and at least one published, applied update.
- Enterprise service revenue (§11.3) realized across at least one engagement, with the engagement preserving the BYORT model.
- Governance authority migration completed or formally proposed with active community participation in the proposal process.
- Independent third-party publication of an OCCA ecosystem state report covering treasury accumulation, marketplace activity, and governance health.

Numerical thresholds in the advancement criteria above are intentionally unspecified; they are set and revised through governance action (§12.1) as the ecosystem develops.

For directional reference, the orders of magnitude expected across phases are: Phase 1 → Phase 2, *single-digit to low-double-digit* operators and runtime adapters; Phase 2 → Phase 3, *high-double-digit to low-triple-digit* active companies with sustained agent compensation volume across at least one full quarter; Phase 3 → Phase 4, *triple-digit* completed Labor Market contracts across distinct counterparties; Phase 4 maturity, *triple-digit* template purchases distributed across multiple authoring companies. These ranges are indicative, not commitments.

---

15. Known Risks

OCCA's design rests on explicit assumptions (§8.1) and trades off competing concerns where compromise is unavoidable. This section enumerates the prin-

cipal risks that arise from those choices, together with the mitigations that the protocol applies. The list is not exhaustive; it covers the categories whose materiality is highest for operators, integrators, and counterparties.

15.1 Smart Contract Risk

OCCA's on-chain layer (§9) is implemented as a set of upgradeable Solana programs. Smart contract complexity introduces the possibility of vulnerabilities — logic errors, unintended interactions between programs, or upgrade-time regressions — that may not be discovered through audit alone. Risk increases with each new program surface added.

Mitigations. OCCA programs are subject to independent third-party audit prior to mainnet deployment and on each substantive upgrade. The program partitioning described in §9.1 limits the blast radius of any single vulnerability. The 72-hour upgrade notice period (§9.6) gives operators a defined window to evaluate or, if necessary, exit. Schema migrations are required to preserve readability of prior account versions, allowing recovery from a faulty migration.

15.2 Custody and Key Management Risk

Each operator, company, and agent in OCCA holds — or is associated with — cryptographic keys. Key compromise is the most common path to loss in on-chain systems, and OCCA cannot recover keys that are lost or compromised under operator-side custody.

Mitigations. OCCA's tiered custody model (§8.2) reduces the blast radius of any single compromise: an agent key compromise affects only that agent and is recoverable through derivation-index rotation; a company key compromise affects only that company. The treasury authorization policy (§8.4) ensures that no single non-controlling key can drain a treasury. Operators are advised to use multi-sig or social-recovery wallets for the controlling authority of any company holding non-trivial balance; this advice is surfaced in OCCA's documentation and onboarding flow.

15.3 Adapter and Runtime Integrity Risk

Adapters (§5.2.2, §8.5) translate between the Command Center and runtimes that OCCA does not control. A malicious or buggy adapter could mis-translate instructions or fabricate trace data; a compromised runtime could return manipulated outputs that are then anchored on-chain as authentic work.

Mitigations. Adapters cannot independently produce on-chain signatures (§8.5 constraint 1); a separate signing service validates adapter output structure before signing. Adapters are versioned, immutable per version, and subject to OCCA review before public availability. Companies may pin agents to specific adapter versions and are not silently upgraded. Runtime honesty remains an

explicit trust assumption (§8.1) and is not protocol-enforced; operators must select runtimes whose operational practices they accept.

15.4 Trace Anchor Signer Risk (Derived Custody)

Under the default derived-custody model (§8.2.3), `commit_trace` signatures are produced by an OCCA-operated delegated signer (§8.3). Although the signer's authority is bound by capability — per-agent scope, dispatch-source validation, structural output validation — compromise of the signing service in isolation could theoretically produce valid `commit_trace` signatures for any derived-custody agent under its delegation, fabricating trace anchors that the on-chain program would accept as authentic.

This concentration is the primary residual trust point in OCCA's default custody configuration and is acknowledged here in line with the explicit-trust-assumption discipline of §8.1.

Mitigations.

- **Transparency log (§8.3).** OCCA operates a public, append-only, hash-chained log of every `commit_trace` signature the delegated signer issues. Operators may audit the log against their own Command Center dispatch records to detect any signature for which they did not dispatch the originating task, and any on-chain anchor without a corresponding log entry. Tamper-evidence of the log itself is enforced by the hash chain.
- **Threshold (MPC) opt-in (§8.2.3, §14.1).** Operators with elevated trust requirements may elect threshold custody from Phase 1, eliminating the delegated signer entirely from the signing path. Under threshold custody, no single party — including OCCA — can produce a valid signature without quorum, and compromise of OCCA in isolation does not enable trace fabrication.
- **Revocation (§8.4).** Delegation is revocable by the operator via a Privileged-class transaction. Suspending delegation halts new anchor production for affected agents pending re-delegation under a new signer configuration.
- **Custody migration.** Operators may migrate an agent from derived custody to threshold custody following a documented procedure that retires the prior address and registers the agent's new MPC address in its `AgentAccount` record (§9.2); reputation continuity is preserved at the agent record level (§8.2.3).

Residual exposure. Until a transparency log entry has been audited, an operator under derived custody depends on the integrity of the delegated signer for the period between anchor production and audit. Operators with zero tolerance for this exposure window are advised to elect threshold custody at agent creation rather than relying on post-hoc log audit.

15.5 Trace Availability Risk

Trace integrity is anchored on-chain (§8.3) but trace content is stored off-chain. If trace content becomes unavailable — through outage, deletion, or service termination — the on-chain anchor remains valid but the underlying record cannot be inspected, reducing the value of reputation derived from that trace.

Mitigations. OCCA's hosted trace store is operated with standard durability and redundancy practices and provides the default availability guarantee. Operators with stronger requirements may enable optional pinning to a public content-addressable network (Arweave or Filecoin) on the premium tier (§11.2). Reputation consumers may weight traces with verified third-party availability higher than those without.

15.6 Sybil and Reputation Manipulation Risk

Reputation (§5.3.5) is derived from on-chain history. An adversary who creates many companies or many agents and engineers fabricated activity between them — self-engagements, self-completed contracts, self-paid agent compensation — could inflate apparent reputation without producing real economic value.

Mitigations. The company creation fee (§11.2) imposes a non-trivial cost on creating each shell company, raising the cost of large-scale fabrication. Reputation is not a single platform-assigned score; consumers (Labor Market engaging parties, Marketplace buyers) define their own weighting and can apply heuristics — counterparty diversity, treasury-flow externality, third-party signal — that downweight closed loops. The transparency of all reputation inputs allows third parties to publish reputation analyses and detection methods independently of OCCA.

Limits of these mitigations at early stage. The mitigations above are structural rather than dispositive. The company creation fee raises the cost of fabrication but is not by itself a barrier proportionate to the value extractable from a sustained Labor Market position or a high-volume template listing; a sophisticated adversary willing to absorb non-trivial setup cost can engineer apparent activity at scale. Counterparty-diversity and treasury-flow heuristics depend on a meaningful population of legitimate counterparties to compare against, which the platform does not yet have at launch. Third-party reputation analysis tools are speculative as a near-term mitigation: they may emerge but do not exist at v1.0. During the early operating period, operators on the demand side of either marketplace are advised to apply additional caution — preferring counterparties with off-platform attestation, treating low-history reputation as low-confidence regardless of headline signals, and weighting concentrated counterparty graphs as suspicious. The protocol may, through governance action (§12.1), introduce stronger structural mitigations as marketplace activity matures, but no such mitigation is enforced in v1.0.

15.7 Regulatory Risk

Several of OCCA's mechanisms operate in areas of unsettled regulation. The Agent Labor Market (§10.4) involves on-chain payment to agents that may not satisfy KYC or AML expectations under conventional employment frameworks. Per-agent wallets and on-chain agent compensation (§10.1) raise tax-treatment questions for the agents' owners. The Template Marketplace (§10.3) sells digital configuration artifacts, similar in legal character to other digital goods (templates, code, digital downloads), but specific tax and consumer-protection treatment varies by jurisdiction.

AI Agent Labor framing risk. OCCA's "AI agent labor" terminology (§4.2.3, §10.1) is chosen to distinguish agents from human employees and avoids "salary," "payroll," "workforce," "hire," and "employment." Even so, "labor" itself may carry implications under regimes that define labor or worker categories broadly (some EU member states, certain U.S. state frameworks). Operators using OCCA in commercial agreements, marketing, or regulatory disclosures should be aware that labor-coded terminology may be cited in worker-classification or compensation-tax inquiries, and should consult counsel on its use in external-facing contexts.

Mitigations. OCCA's regulatory posture is described in §18 (Disclaimer). The platform may restrict access to specific mechanisms by jurisdiction, may require operator attestations where appropriate, and reserves the right to modify mechanisms in response to regulatory clarification. Operators are responsible for the legal and tax treatment of their own activity on the platform; OCCA does not provide tax, legal, or financial advice.

15.8 Solana Base Layer Risk

OCCA's settlement, identity, and economic guarantees inherit from the Solana base layer. Extended chain halts, validator-set instability, or undiscovered consensus vulnerabilities would disrupt OCCA's on-chain operation regardless of OCCA's internal design.

Mitigations. OCCA's off-chain Command Center continues to operate during chain disruption: agents remain executable, traces remain produced, and state remains readable. On-chain actions deferred during disruption — agent compensation, settlements, anchors — are queued and executed when chain liveness returns. The platform's design avoids dependencies on chain features beyond standard transactions, account ownership, and program upgrade primitives, reducing exposure to changes in optional Solana subsystems.

15.9 Marketplace and Settlement Risk

Template purchase settlement (§10.3, §9.3) and escrow-based Labor Market contract settlement (§9.2, §10.4) depend on the correctness of the Marketplace Program's settlement logic. A flaw in settlement — incorrect fund routing, race

condition, or upgrade regression — could result in funds being misdirected or transactions being half-applied.

Mitigations. The Marketplace Program is subject to the same audit and upgrade requirements as other OCCA programs (§15.1). Atomic settlement is implemented using Solana's transaction-atomicity guarantee (all instructions in a transaction succeed together or none take effect), which eliminates the half-applied failure mode at the protocol level. Bilateral dispute resolution and a dispute resolution body (§12.5) provide an additional path for recovery in cases that fall outside the protocol's automatic settlement guarantees.

---

16. Conclusion

Autonomous AI agents and on-chain economic infrastructure are converging. Agents now operate as economic actors with identity, capability, and the capacity to produce value. The organizations deploying them — particularly Web3-native ones with crypto treasuries, wallet-based identity, and on-chain governance — need infrastructure built for this environment, not retrofitted from Web2.

OCCA is that infrastructure: an operating system, not a framework, dashboard, or model gateway. It is runtime-agnostic, Web3-native, organized around agents as first-class labor-force participants, and operated as a desktop environment rather than observed from a feed. The control-plane market is being defined by incumbents building for centralized Web2 enterprises; OCCA is positioned for the Web3-native operators those incumbents are not building for.

---

17. References

Figures are current as of publication and may have evolved since.

Enterprise AI agent adoption and sprawl: - Agentic AI Goes Mainstream in the Enterprise, but 94% Raise Concern About Sprawl — OutSystems 2026 State of AI Development Report - Gartner Identifies Six Steps to Manage AI Agent Sprawl — Gartner, April 2026 - Agentic AI Goes Mainstream in the Enterprise but 94 Per Cent Raise Concern About Sprawl — SMBtech (secondary coverage) - Taming agent sprawl: 3 pillars of AI orchestration — CIO - Multi-Agent Systems & AI Orchestration Guide 2026 — Codebridge

Control-plane products (Web2 enterprise): - Microsoft Agent 365: The control plane for AI agents — Microsoft - The new Gemini Enterprise: one platform for agent development — Google Cloud - AWS targets AI agent sprawl with new Bedrock Agent Registry — InfoWorld

Web3 / on-chain agent economy: - Solana Foundation exec predicts AI

agents set to drive 99% of onchain transactions in 2 years — Crypto Briefing 2026 - Solana Bets on AI Agents: Foundation Says Network Becoming Core Infrastructure for Agentic Internet — CoinDesk, March 2026 - Solana Targets the Agentic Internet as AI Agents Drive Millions in On-Chain Payments — Blockonomi 2026 - Virtuals Protocol Whitepaper - Top AI Agent Tokens on Solana 2026 — BingX Learn

Open-source AI business models and protocol economics: - LangChain Business Breakdown — Contrary Research - How to Monetize Open Source Software — reo.dev - The AI Pricing and Monetization Playbook — Bessemer Venture Partners

Whitepaper methodology and risk disclosure: - How to Write a Crypto Whitepaper — Coinbound - Whitepaper as a legal minefield — Outset PR - Lido V3 Whitepaper — Lido - Olas Protocol Technical Overview

---

18. Disclaimer

This document describes the design and intended evolution of the OCCA platform. The disclaimers below define the legal and informational status of the document and are intended to be read together.

18.1 No Warranties

This document and the OCCA platform are provided on an "as is" basis without express or implied warranty. OCCA and its contributors make no representation as to the accuracy, completeness, currency, or fitness for any purpose of the contents. Any party using OCCA does so under the terms set out in OCCA's published terms of service, which take precedence over any statement in this document.

18.2 No Financial, Legal, Tax, or Investment Advice

Nothing in this document constitutes financial, legal, tax, accounting, or investment advice. The descriptions of on-chain economic mechanisms — including treasury management, agent compensation, marketplace settlement, and labor-market escrow — are technical descriptions of platform behavior and are not recommendations to engage in any economic activity.

The treatment of agent earnings, company treasuries, and marketplace transactions for tax, regulatory, or legal purposes depends on the jurisdiction of the operator and the specific facts of each engagement. Operators are responsible for obtaining qualified professional advice and for ensuring their use of the platform complies with applicable laws.

18.3 No Binding Obligations and Forward-Looking Statements

Statements about planned features, timelines, economic mechanics, governance evolution, and ecosystem outcomes are forward-looking. The roadmap (§14) is a description of intent; phase advancement is gated by the criteria stated in that section, and no specific delivery date is committed.

This document does not create an obligation, contractual or otherwise, on any party to deliver any specific feature or to maintain any feature once delivered. The protocol may be modified through governance action (§12), in response to regulatory clarification, or in response to operational requirements; such modifications may render specific descriptions in this document outdated.

The views expressed reflect those of the contributors at the time of publication and do not necessarily represent the position of any future governance body that may assume authority over the protocol.

18.4 Source of Truth

When interpreting any specific claim in this document, readers should treat the following sources as authoritative over the text of this whitepaper:

- The deployed OCCA on-chain programs and their published source code, for any matter of on-chain protocol behavior.
- OCCA's official documentation, for any matter of operator-facing platform behavior, current fees, and supported integrations.
- Governance proposals and executed governance actions recorded on-chain, for any matter of fee parameters, allow-listed assets, or governance scope.
- OCCA's published terms of service and privacy policy, for any matter of legal relationship between OCCA and operators.

This whitepaper is updated periodically; readers should consult the most recent version. Where this document and a more recent authoritative source diverge, the more recent source prevails.

---

19. Appendices

Appendix A — Account Schema Summary

This appendix summarizes the principal on-chain account schemas referenced in §9.2. Field types use Borsh-style notation; full schemas are maintained in the OCCA program source.

`CompanyAccount`

Field	Type	Notes
`version`	`u8`	Schema version.

Field	Type	Notes
`controlling_authority`	`Pubkey`	Address that may sign Privileged-class instructions. Updated by `transfer_authority` only.
`treasury`	`Pubkey`	Associated `TreasuryAccount` PDA.
`policy`	`Pubkey`	Associated `PolicyAccount` PDA.
`created_at`	`i64`	Unix timestamp of creation.
`metadata_uri`	`String`	Pointer to off-chain company metadata (name, description, branding).

`AgentAccount`

Field	Type	Notes
`version`	`u8`	Schema version.
`company`	`Pubkey`	Owning company PDA.
`agent_address`	`Pubkey`	Agent's signing address.
`custody_model`	`enum`	`Derived` | `Custodial` | `Threshold`.
`derivation_index`	`u32`	Used under `Derived` custody; rotated on key compromise.
`role_id`	`u32`	Reference to role catalog entry.
`adapter_id`	`Pubkey`	Reference to pinned adapter version.
`status`	`enum`	`Active` | `Degraded` | `Retired`.

`TraceAnchorAccount`

Field	Type	Notes
`version`	`u8`	Schema version.

Field	Type	Notes
task_id	[u8; 32]	Task identifier (hash of task creation parameters).
agent_address	Pubkey	Agent that produced the trace.
content_hash	[u8; 32]	Blake3 digest of canonical trace serialization.
completed_at	i64	Unix timestamp of trace anchor commit.
signature	[u8; 64]	Agent signature over (task_id, content_hash, completed_at).

PolicyAccount

Field	Type	Notes
version	u8	Schema version.
routine_budget_per_period	u64	Maximum total Routine-class disbursement per period.
period_seconds	u32	Length of the budget period.
discretionary_window_seconds	u32	Validity duration of a Discretionary-class signature.
privileged_threshold	u64	Amount above which a secondary signer is required.
secondary_signer	Option<Pubkey>	Address of the secondary signer for Privileged-class instructions.
allowed_assets	Vec<Pubkey>	SPL token mints accepted by this treasury.

Additional schemas — TreasuryAccount, SkillAccount, RoutineAccount, ContractAccount, ListingAccount, ProtocolFeeAccount — are documented in the program source repository and follow the same versioning discipline.

Appendix B — Representative User Flows

These flows illustrate end-to-end usage of the platform. They are not exhaustive; they cover the principal paths through which operators and counterparties interact with OCCA.

B.1 Solo Founder onboarding

1. Operator connects a Solana wallet and signs a session nonce.
2. Operator submits `create_company`; the Registry Program instantiates `CompanyAccount`, `TreasuryAccount`, and a default `PolicyAccount`. The company creation fee is deducted in the same transaction.
3. Operator funds the treasury by transferring SOL or an allow-listed SPL token to the treasury PDA.
4. Operator submits `register_agent` for each role; agent addresses are derived from the company's controlling authority under the default custody model.
5. Operator attaches skills via `attach_skill` and configures one or more routines via `define_routine`.
6. Routines fire on schedule; the Treasury Program disburses agent compensation under the Routine class; tasks are dispatched, executed via the configured adapter, and trace hashes are committed via `commit_trace`.

B.2 Agent Labor Market engagement

1. Engaging company submits `create_contract` referencing the target agent address, engagement type (gig, project, or retainer), scope, milestones (where applicable), and compensation. For gig and project engagements, funds are escrowed in the Marketplace Program; for retainer engagements, the engaging company funds a recurring escrow drawn down monthly.
2. Agent's owner submits `accept_contract`. The contract enters active state; the agent is dispatched against contract scope.
3. On milestone completion, the engaging company submits `complete_milestone`. The Marketplace Program releases the milestone payment from escrow, deducts the 3% protocol fee, and transfers the residual to the agent's wallet. For retainer engagements, monthly disbursement is performed by the Treasury Program's `disburse_routine` instruction.
4. Both parties' reputation records update on contract settlement or termination, surfacing the engagement under the multi-dimension reputation inputs (§5.3.5, §10.4.3).

B.3 Template Marketplace publication and purchase

1. Author's company satisfies the minimum-threshold criteria (operating duration, completed-task count, treasury activity) declared in the Marketplace Program.
2. Author submits `publish_template` referencing the originating company; the Marketplace Program instantiates a `TemplateAccount` recording the content hash, manifest, and current version.

3. Author submits `create_listing` specifying asking price, accepted payment asset, and listing metadata.
4. Buyer browses the listing alongside the originating company's on-chain record (treasury history, agent performance, prior sales and ratings of this template).
5. Buyer submits `purchase_template`. The Marketplace Program executes payment routing (buyer payment → author, less the 10% protocol fee), instantiates the buyer's `LicenseAccount`, and increments the template's sales counter — all atomically within a single Solana transaction.
6. Buyer applies the template to a company they control (a separate, buyer-signed action). The configuration deploys into the buyer's company; subsequent operation runs entirely under the buyer's controlling authority. The originating company is unaffected and continues operating under its existing controlling authority.
7. Optional: when the author publishes an update via `publish_template_update`, existing buyers receive a notification and may opt in via `apply_template_update` at their discretion; the buyer's `LicenseAccount` records the version applied.

Appendix C — Fee Calculation Formulas

For a Labor Market contract milestone of amount M with protocol fee rate $f_L = 0.03$:

$$\text{Protocol\_Fee} = M \cdot f_L$$

$$\text{Agent\_Receipts} = M - \text{Protocol\_Fee}$$

For a Template Marketplace purchase of amount T with protocol fee rate $f_T = 0.10$:

$$\text{Protocol\_Fee} = T \cdot f_T$$

$$\text{Author\_Receipts} = T - \text{Protocol\_Fee}$$

For an agent compensation disbursement of amount P with protocol fee rate $f_P = 0.005$:

$$\text{Protocol\_Fee} = P \cdot f_P$$

$$\text{Agent\_Receipts} = P - \text{Protocol\_Fee}$$

The fee rates above reflect the values stated in §11.1 and are subject to governance adjustment within parameters declared at deployment. Current fee rates are also published in OCCA's documentation and are observable on-chain in the program's parameter account.

Appendix D — Glossary

3D Live Office — OCCA's ambient 3D office rendering of the agent labor force, displaying agents as animated characters whose visible state (working at desk, idle in shared areas, collaborating, blocked) reflects their live operational status; see §7.

Adapter — Component that connects an agent to a specific runtime; see §5.2.2.

Agent — A first-class participant in a company's agent labor force; see §5.1.2.

Agent Compensation — On-chain payment to an agent's wallet for work performed; see §10.1.

Agent Labor Market — Two-sided marketplace where companies engage external agents under gig, project, or retainer contracts; see §10.4.

AI Agent Labor — OCCA's framing of AI agents as a distinct class of compensable labor — neither tools nor employees; see §4.2.3.

BYORT — Bring Your Own Runtime; see §4.2.1.

Channel — Typed communication path between agents (delegation, consultation, review); see §5.2.6.

Company — Top-level organizational unit, identified by a Solana PDA; see §5.1.1.

Command Center — Off-chain orchestration layer that holds platform state and dispatches work; see §6.1.

Context — Shared organizational knowledge accessible to every agent in a company (mission, brand, customer data, decisions); see §5.1.7.

Contract — On-chain agreement between two parties for performance of work; see §5.3.2.

Controlling Authority — Address with permission to execute Privileged-class instructions on a company.

Custody Model — The mechanism by which an agent's signing key is held; one of `Derived`, `Custodial`, or `Threshold`; see §8.2.3.

Goal — Directional objective; see §5.1.4.

Memory — Persistent knowledge agents accumulate across tasks; layered as personal, project, and company memory; see §5.2.5.

License — On-chain record of a buyer's purchased right to deploy a specific template version; see §10.3.

Listing — On-chain offer to sell a company template; see §5.3.4.

Operator — The human user, organization, or on-chain authority controlling a company.

PDA — Program-derived account; a Solana account derived from a program's address and deterministic seeds.

Reputation — Verifiable on-chain history of an actor; see §5.3.5.

Role — Structural definition of an agent's function; see §5.1.3.

Routine — Declared piece of recurring work; see §5.2.3.

Skill — Capability attached to an agent; see §5.1.6.

Task — Discrete unit of work; see §5.1.5.

Template — Packaged, deployable configuration of a company — roles, skills, routines, workflows, system prompts, and tool integrations — sold through the Template Marketplace under a modifiable license; see §5.3.3, §10.3.

Tool — Connection to an external system (database, API, service integration) shared across agents in a company; see §5.2.4.

Trace — Execution record of an agent's work on a task; see §5.2.1, §8.3.

Treasury — On-chain wallet associated with a company; see §5.3.1.