Autonolas Whitepaper

Autonolas Whitepaper

version 1.0

1. Autonomy

a. Context

Humans have always strived to automate repetitive processes in their everyday lives. Today, as "software is eating the world", the focus is shifting from automation toward higher levels of autonomy. We believe that autonomous services (which run continuously and take their own actions) are a technological imperative in a future with increased human autonomy.

In sharp contrast to our vision, current web-based, global mega-corporations create automation that, more often than not, places humans at the service of machines. Marketing techniques are increasingly brazen in confounding the distinct concepts of autonomy, using "self-operating" (as in machines and systems) which the consumer can take to mean "user-determined". Self-driving cars are an example: "autonomy" is used as a selling point for self-operating systems that are decidedly not user-determined; users yield significant control to opaque algorithms, and forfeit their private data.

Autonolas seeks to maximize machine autonomy to benefit human autonomy. We believe that autonomous services – services that require little to no input from humans by design – are best used in the service of humans. When operated using transparent software by groups, with open participation, the extreme reliability of autonomous services benefits humans the most.

We are fascinated by the potential of increased human autonomy delivered by autonomous products, to wrest control of human agency from systems and machines, and to help reprioritize our lives.

b. Autonomy in Crypto

Crypto is the natural home from which to grow our vision and presents the market need for autonomous products as we define them here. The concept of Decentralized Autonomous Organizations (DAOs) was born in the crypto space[1] and became popular with the rise of Ethereum [2]. However, while blockchains are great for building robust

[1] The name "Decentralized Autonomous Organization" was first mentioned however in the field of cybernetics to describe intelligent home technology as a complex, *multi-agent system of algorithms* that would operate like a living organism [1].

and decentralized applications, they are particularly ill-suited for building truly autonomous products.

DAOs and autonomous applications more generally cannot run solely as on-chain smart contract applications. By design, smart contracts are extremely unsuitable for certain activities, and cannot do the following:

- Do things that require expensive processing.
- Run continuously.[2]
- Access data not on the same blockchain, like external APIs or other chains.

This means that today, most DAOs are neither autonomous nor decentralized. DAOs are thus forced into two mutually-exclusive scenarios:

- They remain entirely on-chain, relying on incentivizing ad-hoc external actors to regularly call their smart contracts. This makes the system unreliable and limits its potential.

- They forgo decentralization by undertaking core DAO operations in a centralized manner off-chain, e.g. by running a bot or manually sending transactions. This suffers from single-operator risks, unauditable off-chain processes, and non-reusable off-chain software.

A few concrete examples of these limitations:

- Off-chain operators must be trusted. These operators are usually run by a single entity, and as a result, some degree of reliability is sacrificed.
- Single-entity off-chain operators are exposed to serious personal liability concerns given the current uncertain regulatory landscape.
- An on-chain protocol cannot manage its social media advertising budget.
- An on-chain asset management fund could not run machine-learning processes to optimize its trading strategies.
- Inter-DAO operations as well as intra-DAO operations are not interoperable or composable. For example, DAOs are incapable of autonomously negotiating and purchasing services from one another.

[2] The Ethereum blockchain core developers did briefly consider including an ALARM opcode to enable smart contracts to schedule operations in future blocks, but it was ultimately discarded as unworkable [https://vitalik.ca/general/2022/03/29/road.html]. The Cosmos SDK used for development of application specific blockchains [https://v1.cosmos.network/sdk] has some support to execute code – with significant limitations – at the beginning and end of each block. For example, it is used by Osmosis in their epoch module. However, the same general limitations apply as discussed in Vitalik Buterin's article.

This indicates a major missing piece of infrastructure for realizing "full-stack autonomy". Furthermore, there is an economic opportunity in coordinating developers, operators, and investors to create, run and fund parts of this infrastructure.

c. Autonolas Proposition

Autonolas' mission is to provide a foundation on which new types of autonomous applications can be built, and which applications can do the following:

- Run continuously.
- Take action on their own.
- Interact with the world outside of blockchains.
- Run complex logic.
- Be composed of self-contained application modules.
- Be crypto-native: decentralized, trust-minimized, transparent, and robust.

In the context of blockchain, we call these applications *off-chain services*, as they run primarily off-chain but are secured on-chain.

Autonolas provides the following:

- The open-source software stack to realize these services.
- An on-chain protocol, eventually deployed on all major smart-contract blockchains, that secures these services and incentivizes their creation.
- Tokenomics designed to attract more and more capital and developers into a virtuous flywheel, in turn increasing the number of autonomous applications.

The Autonolas stack enables building of services made up of multiple agents. Composability at the stack and services layers enables Autonolas to convert linear component, agent, and service growth into exponential application opportunities. Moreover, it goes beyond the usual architecture of software frameworks by allowing developers to encapsulate business logic into software components. As such, we ultimately see this stack used in autonomous applications outside crypto.

d. Why now?

> ***"Autonolas powerful tech stack combines recent crypto technologies and draws on MAS to enable building composable autonomous systems."***

Two key technological factors allow Autonolas to overcome previous limitations. One is the ready availability of production-grade Byzantine fault-tolerant consensus engines like Tendermint. The other factor is insights from decades of Multi-Agent Systems (MAS) research that so far have been largely ignored in crypto.

MAS are computerized systems composed of multiple interacting, equally-privileged agents. Every such agent can directly connect and interact with every other agent without intermediaries. This enables a multi-agent system to be decentralized and more robust than software relying on single agents, bots, or processes. Autonolas powerful tech stack combines recent crypto technologies and draws on MAS to enable building composable autonomous systems.

The recent growth in DAOs and the multi-chain reality suggest now is the right time to take Autonolas to market. Current DAOs are unable to scale without significant infrastructure improvements, and the multi-chain world places increasingly technical integration requirements on project teams. An incentivized open-source infrastructure stack with reusable and composable software addresses both problems head-on.

e. A Glimpse into the Future

Three themes dominate our vision as it plays out in crypto. First, open-source developers contributing to services secured on Autonolas protocol will find a reliable and permissionless source of income for their work. Second, service owners, and in particular DAOs, will be able to realize increasingly user-friendly products that do not compromise on robustness, transparency, and decentralization. Finally, investors of the protocol and its services will see rewards from the utility and efficiency savings driven by their investments.

Further into the future, things are more difficult to predict. However, the history of technology can inform us to a degree. As noted by Carlota Perez [3] and Tim Wu [4], and synthesized by Placeholder in their investment thesis [5], the arc of technology tends to trace cycles of consolidation and dissolution of monopoly power around scarce resources. In any given era, there is one scarce resource.

In the previous era, this was data, and monopolies were built around software services that depended on this data, for example, Google with search data and Facebook with the social graph. Crypto technologies have the potential to dissolve the moat that data monopolies provide.

When this occurs, we assert that it is high-quality full-stack autonomous systems that will become the scarce resource. Powerful organizations will emerge around an entirely new category of value – the ability to buy *full outcomes as a product*.

Some very manual and simple services already resemble this. Buying a travel package from a vertically-integrated travel agency is one example. However, we imagine that networks of self-organizing, fully autonomous entities can massively extend the breadth, quality, and affordability of such services.

f. Whitepaper Outline

This document will outline how Autonolas' vision can be realized concretely. In the 'Technical Architecture' section, we describe 4 key building blocks of the Autonolas tech stack that enable full-stack autonomy:

- the Multi-Agent Systems Architecture
- State-Minimized Consensus Gadgets
- the Architecture of Crypto-native Off-chain Services
- the On-Chain Protocol

Full-stack autonomy will facilitate autonomous off-chain services that, as we shall see, are trust-minimized, robust, and transparent.

In 'Tokenomics', we outline the proposed economic model facilitated by the Autonolas token. This is a mechanism to grow the capital deployed in services in tandem with the useful code that makes up these services. We highlight how the composability of the software stack extends to the tokenomics, and we introduce *the novel concept of Protocol-owned Services*. As we shall see, this facilitates and incentivizes the creation of autonomous services owned by the governance of various DAOs, operated by the Autonolas ecosystem, and developed by agent developers.

In 'Use Cases for Agent Services', we suggest the kinds of autonomous services that the Autonolas tech stack enables. We identify two main areas of application, namely, in DAO operations (examples include smart treasury management and autonomous protocol parameter configuration) and protocol infrastructure (including more complex and custom-made oracles and robust keepers).

In 'Governance', we explain the design of Autonolas as a decentralized, autonomous organization, that is governed by the community and brought to life in the lore universe of Alter Orbis.

Finally, at the very end, you can also find a Glossary to refer to more complex terms, and an Appendix.

2. Technical Architecture

In this section, we describe the main building blocks of the Autonolas tech stack. This autonomous software stack for crypto will be able to, among other things, perform complex processing, take action on its own, and run continuously, none of which can be done by building purely on-chain.

Open Autonomy is Autonolas software framework for the creation of agent services: off-chain autonomous services which run as a multi-agent-system and offer enhanced functionalities on-chain. The Autonolas software stack lives primarily off-chain, where the possibilities are vast and where novel and fully autonomous applications can be realized. However, extraordinarily, Autonolas is not sacrificing on-chain properties of trust-minimization, robustness, and transparency. Instead, Autonolas enables crypto-native autonomous services[3] in the space along the continuum between centralized bots and scripts, and decentralized blockchains.

The Autonolas stack goes beyond the usual monolithic architecture by encapsulating business logic. Composability at the stack and services layers enables Autonolas to convert linear component, agent, and service growth into exponential application opportunities. As such, we ultimately see this stack used in autonomous applications outside crypto.

Below, we explore each of the 4 building blocks of this powerful, composable architecture:

a. Multi-Agent Systems Architecture – the core blueprint for architecting autonomous services

b. Consensus for Agent Services – how we repurpose off-chain consensus engines to enable robust, transparent, and decentralized off-chain consensus

c. Architecture of Crypto-native Off-chain Services – the current architecture of autonomous services as agent services

d. On-Chain Protocol – the means by which we secure autonomous services running off-chain

[3] Vitalik Buterin speaks of the “missing middle” [https://twitter.com/VitalikButerin/status/1479815125955715072] which Autonolas squarely occupies and defines. In fact, Autonolas expands the “missing middle” as defined in that tweet by introducing off-chain capabilities which have on-chain-like properties but expand the possible space of applications which crucially also can affect the user experience significantly.

The entire stack is required to operate autonomous services and achieve full-stack autonomy.

a. Multi-Agent Systems Architecture for Agent Services

Autonolas' approach to building autonomous software services is to harness Multi-Agent Systems (MAS). MAS are groups of software agents deployed in adversarial and multi-stakeholder scenarios with competing incentives. As such, we will refer to autonomous services as agent services for the remainder of this section.

Today, the primary means to implement an Autonolas agent service is by using (to our knowledge) the only open-source and crypto-friendly MAS framework, called the open-aea framework [6], co-created by an Autonolas co-founder.[4]

In the open-aea framework, an Autonomous Economic Agent (AEA or agent for short) can be seen as autonomous software belonging to an organization, individual, or another agent. The AEA perceives its environment and acts autonomously, pursuing predetermined goals for its owner.

The current implementation of the open-aea framework is the open-aea library [6]. open-aea is a Python-based open-source application library that enables developers to create agents[5] with a particular application focus on decentralized ledger technology (DLT)[6]. More concretely, it allows developers to equip agents with the ability to:

- Communicate peer-to-peer
- Interact directly with blockchains (such as Ethereum) and their smart contracts
- Implement arbitrary business logic for reuse through components

Agents are made up of reusable components[7] that the community can develop and use to create more complex agents or agent-based applications. Specifically, a

[4] The legacy AEA framework was co-created by David Minarsch during his tenure at Fetch.ai. The open-aea framework evolves the legacy AEA framework and removes any vendor tie-ins. It is the dominant framework for implementing agent services. However, we envision many complementary frameworks fulfilling this role in the future.

[5] Here, the term *agents* is to be understood as *canonical agents* defined in the Section Elements.

[6] For the purposes of clarity in this paper we will only address the most commonly known subset of DLT, blockchain technology, even though Autonolas technology can be applied far beyond it.

[7] Here, the term *components* is to be understood as *agent components,* as defined in the Section Elements.

developer can both create an agent by combining already-developed components, and develop their own components, tailored to specific needs.

Reusable components provided by the open-aea include the following:

- Protocols: defining agent-to-agent interactions, and component-to-component interactions within agents
- Connections: interfaces for the agent to connect with other agents and external APIs
- Contracts: wrap logic to enable interaction with smart contracts of blockchain apps
- Skills: self-contained capabilities that the agent can adopt as needed in different situations

Although currently the framework is implemented in Python, agents and their components can be developed in arbitrary programming languages and utilizing arbitrary frameworks, as long as they satisfy a number of specific technical requirements enabling their interoperability (e.g. correct implementation of protocols and application logic).

i. Agent Development and Extensibility

A typical agent software development process involves creating agent components as so-called packages. Most of the efforts are likely to be devoted to the development of Skills, although occasionally there might be the need to create tailored Protocols, Connections, or Contracts.

With this in mind, the framework facilitates the developer process by providing mechanisms for generating scaffolds, for example, for Skills. These can be subclassed from basic component classes, and refined to adapt to the developer's needs.

The Autonolas development process enables the extensibility and reusability of components to create custom agents for specific needs, in particular, as part of services. Each agent, or agent component, defines a YAML configuration file that specifies both its sub-components and its component dependencies. The configuration file also defines a collection of parameters required for the different components, such as the frequency interval of some repeated action. In the case of the agent, it also points to the agent-level configuration. This can include information like the details of the agent's wallet which controls the agent's funds on various blockchains and signs transactions or messages.

b. Consensus for Agent Services

Agent services built in the Open Autonomy framework are multi-stakeholder systems, where individual agents[8] work collectively toward a common goal, implementing the logically centralized application state of the service and that use a distributed ledger as a settlement layer. This group of agents must collectively agree on the next action, or on the output of the computation they are performing. An agent service thus belongs to the category of distributed systems.

Consensus mechanisms reside at the heart of distributed systems (e.g. distributed ledgers) providing a shared state and an objective view agreed upon by multiple participants. Below, we summarize Autonolas' approach to building agreement among agents within a collection without resorting to a full-blown blockchain.

Today, the most well-known permissionless blockchains are built from crypto-economic consensus mechanisms such as Proof-of-Work (e.g. Bitcoin, Ethereum) and Proof-of-Stake (e.g. Ethereum 2.0, Cosmos). Permissioned blockchains mostly use Proof-of-Authority consensus mechanisms (e.g. Binance Smart Chain, Hyperledger), where blocks are validated by a *strategically selected* group of authorities that can be voted out in case of misbehavior.

However, in the rapidly growing ecosystem of distributed ledger technologies, an increasing diversity of consensus mechanisms has emerged with the goal of adding certain features [7]. These include:

- Scalability: handling a growing amount of work
- Consistency: enabling honest nodes in the system to agree upon the same view of the current state
- Security: ensuring state history cannot be reverted
- Fault-tolerance: continuing to operate regardless of failures or malfunctions of some of the nodes

i. State-minimized Consensus Gadgets: Secure, Robust, Efficient and Trust-minimized

The Autonolas stack enables the creation of *secure*, *robust,* and *trust-minimized* agent services. We call the key ingredient *state-minimized consensus gadgets* (or "consensus gadgets").

[8] Here, the term *agents* is to be understood as *agent instances* as defined in the Section Elements.

Roughly speaking, a consensus gadget implements a protocol that allows a set of agents, which make up an off-chain agent service, to reach agreement on the value of one or more variables. We refer to this as achieving consensus internal to the agent service. We next describe the properties of security, robustness, and trust-minimization for consensus gadgets.

Secure

How do consensus gadgets work? Similar to many PoS-based consensus engines, agent services' internal consensus is obtained after a finite sequence of rounds. The transition from each round to the next occurs when certain conditions are met. When an agreement is reached, consensus is recorded locally in the agent service.

The agent service may choose to record snapshots of parts of that state on an L1/L2 blockchain ledger of the service owner's choice. An agent service will typically compute functions of the intermediate values of the local state, and record them on-chain, that is in the given L1/L2 chain.

As a blockchain-oriented architecture, Autonolas chooses to rely on a consensus gadget that in the first instance allows *deterministic finality.* When consensus is reached and recorded, the state is final and cannot be changed.

This is one of two reasons why we prefer to build from Proof-of-Stake over consensus engines based on Proof-of-Work. With the former, it is possible to reach *deterministic* finality; the latter only allows *probabilistic* finality.

Furthermore, PoS can be complemented by combining it with an algorithm for reaching Byzantine consensus among the parties making up the various services. For example, Tendermint is a practical Byzantine fault-tolerant algorithm that uses PoS as its validator set selection method.

Robust

To unlock autonomous applications, machines must be able to work together as a group such that the group can remain functional even if some of its members fail. Thus, consensus gadgets approximate the fault-tolerance guarantees that power robust L1 chains.

Efficient and Trust-minimized

A crucial advantage of consensus gadgets over L1 consensus mechanisms is that agent services do not need to permanently keep the history of the updated state. Only the relevant states are agreed upon in a consensus gadget and a subset of them get recorded on the L1/L2 chain where the agent service is registered. The economic security of agent services is bootstrapped from the chain where the service is defined and the relevant service states recorded.

Once the state agreed at the consensus gadget level is no longer relevant to the off-chain agent service, it can be deleted. This implies that agent services may be internally using blockchain-like technology, but they do not need to implement a standard L1/L2 layer. Agents can punish each other's misbehavior by submitting fraud proofs to the underlying chain, causing slashing. The agent service retains a reasonable level of decentralization by minimizing the trust placed on individual agents.

Limiting the size of the internal state that the consensus gadget replicates lowers the cost and energy impact of agent services. The frequency with which the internal recorded state can be deleted depends on the needs of the service.

ii. An Initial Implementation of Consensus Gadgets

In an initial implementation, we leverage Tendermint [8] as a consensus engine. It is Byzantine fault-tolerant (BFT), securely and consistently replicating an application on many machines that may fail in arbitrary ways, including malicious faults. Tendermint powers fully blown L1 chains, such as those in the Cosmos ecosystem. As such, it is to some extent more powerful and has a different scope than the consensus gadgets proper to Autonolas.

Tendermint's Application BlockChain Interface (ABCI) allows for BFT replication of applications written in any programming language. In the context of Autonolas, ABCI apps have two salient aspects.

First, ABCI is an interface that defines the boundary between the consensus engine (the blockchain) and the state machine (the application) using a client-server architecture. This enables the consensus engine to run in one process while it manages an application state running in another process. Second, ABCI applications are *only reactive*, as they are limited to specifying how the application state is updated when a transaction is received.

Autonolas innovates in the second aspect by implementing ABCI applications that are *proactive* (instead of reactive), e.g. periodically executing some routines or calling external APIs.

Autonolas builds on the first aspect and equips consensus gadgets with two types of interaction, both following a client-server paradigm:

- **Agent–Consensus Node:** The consensus node (the server) listens for requests coming from the agent (the client). For example, an agent sends a request to notify the node that a transaction needs to be validated.
- **Consensus Node–ABCI Application:** The ABCI application (the server) listens for requests coming from the consensus node (the client). For example, a node sends a request to notify the app that a block has been validated.

The setup of the initial implementation for consensus gadgets is as follows:

- A MAS-based service consisting of a pool of n agents.
- Each agent runs a corresponding Tendermint node, with a total of n Tendermint nodes operating the consensus gadget.
- One or more target L1/L2 blockchains to be used as a settlement layer.

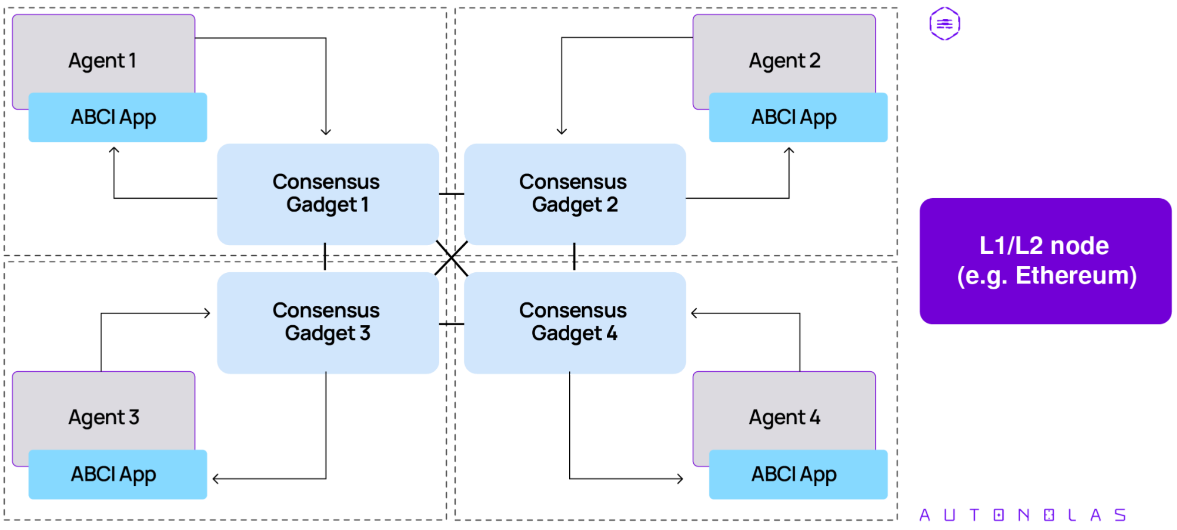

Fig. 1: Illustration of Agent to Node and Node to ABCI app connections with Ethereum as the L1 settlement layer.

Reliable autonomy results from the interplay of the Consensus Node, the ABCI application, and the Agent. The former two provide the fault-tolerant replication, and the latter provides the proactivity missing from traditional blockchains.

iii. Alternative Consensus Gadget Implementations

The initial implementation is built with Tendermint, but other consensus engines can equally be repurposed to build Consensus Gadgets. We discuss them below.

Snowflake

Snowflake [9] is the consensus engine used in Avalanche's L1 chain. A crucial property of Snowflake is that it allows a large network to reach an approximately final state quickly and efficiently. Although Snowflake does not have deterministic finality, the probability of reverting transactions can be made arbitrarily small. Building Snowflake-based Autonolas consensus gadgets may apply to agent services that need a large number of agent operators.

Solana

Solana's consensus protocol [10] enjoys a high transaction throughput at the expense of a somewhat higher centralization. Building consensus gadgets on top of Solana's consensus seems particularly attractive whenever the agent service needs high throughput and the number of agent operators is small.

Lachesis

Lachesis [11] is a DAG-based[9] consensus protocol with a low time-to-finality for transactions, where every confirmed transaction is final, under the assumption that at least 2/3 of validators are honest[10]. Lachesis is the consensus engine used in Fantom's L1 chain. As with Tendermint, Fantom integrated the ABCI functionality, providing a clean interface between any finite state machine on a computer and the mechanisms of a blockchain-based replication engine across multiple computers (consensus engine). As such it is a potential alternative to Tendermint as the basis of consensus gadgets.

c. Architecture of Crypto-Native Agent Services

Off-chain agent services are logically centralized applications that are replicated across multiple machines, controlled by distinct economic actors. They are therefore both distributed and decentralized. Autonolas' architecture for agent services leverages the use of a temporary blockchain created using consensus gadgets.

An *operator* (see Fig. 2) is in control of one agent and one consensus gadget node. These two piecies are interconnected as two processes communicating within the same system using Unix sockets; or as two processes implemented at different machines, communicating via a network. The nodes are in charge of receiving client calls from the agents and forwarding ABCI callbacks to the agents to be handled appropriately.

Each agent runs an ABCI application instance which defines a finite-state machine (FSM) that is replicated on the temporary blockchain. The states of the FSM are called *rounds* and each leads to the agreement on the temporary blockchain where the internal consensus is recorded.

[9] DAG stands for Directed Acyclic Graph, and it refers to a data structure with partial ordering.

[10] More precisely, at least 2W/3 of the validators are honest, where W is the total validating power in a weighted DAG. See [11] for more details.

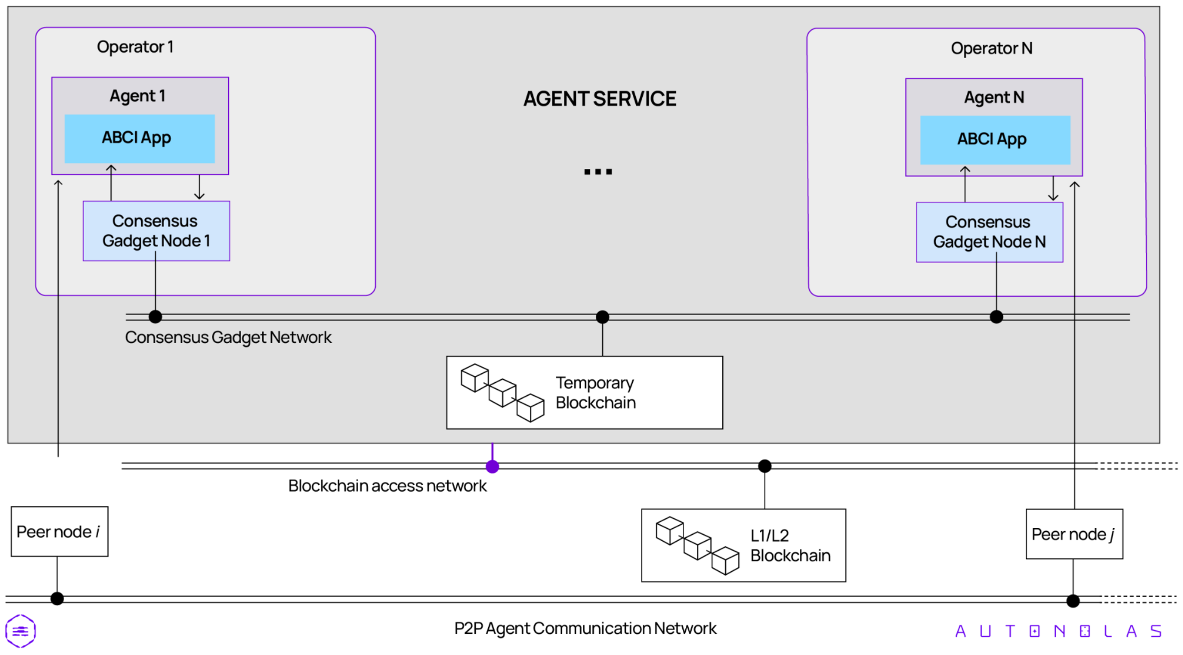

Fig. 2: Overview of the Agent Services Architecture.

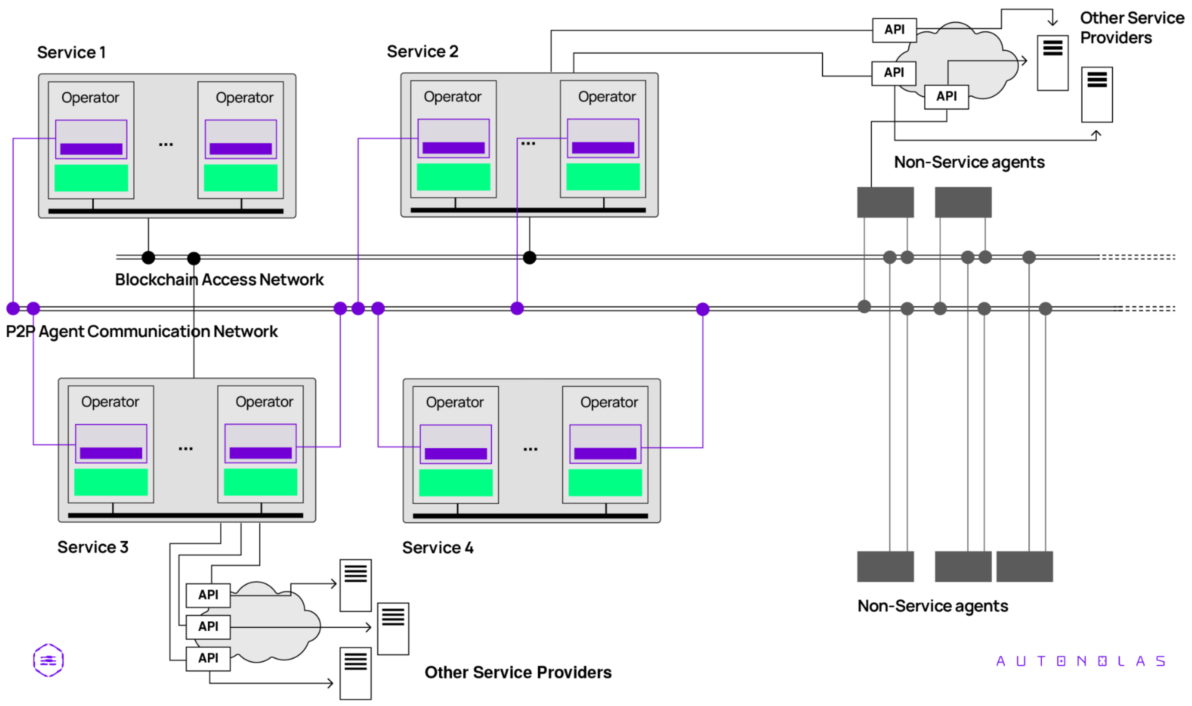

Fig. 3: Putting the Agent Services Architecture into context with the Web.

Each round has its own business logic that specifies how the agents' transactions are validated or the conditions that trigger a transition to another round. The transitions are triggered by means of *events*, that is *values* that are set by the round implementation and are processed by the agent framework to compute the round successor.

The events are of two kinds: *normal* events and *timeout* events. Normal events are set by the round according to certain conditions evaluated each time a block is confirmed by the consensus engine and delivered to the ABCI application. Timeout events are automatically triggered in case the timeout associated with the event has expired.

A behavior called *FSMBehaviour* allows agents to perform state updates of the ABCI application by committing transactions on the temporary blockchain. FSMBehaviour utilizes the ABCI application as a state-replication layer among different agent participants. In contrast with the purely reactive nature of the ABCI application, the agents' FSMBehaviour shows a proactive logic.

For example, each agent's FSMBehaviour can observe the price value of BTC from an external source. The agents can then use the ABCI application to replicate the observations and establish an estimate. Finally, one of the agents can submit the estimate to the given L1/L2 blockchain.

In each step, agents reach consensus on their joint actions using the consensus gadget. The agents never hold any funds of the service, acting on them only via a multisig (at the service level). An individual agent holds only the funds it needs to pay for gas fees to submit transactions (i.e. only the native L1/L2 currency for gas payments).

To enable the composition of services from constituent and reusable parts, Autonolas supports the chaining of ABCI applications. The composition of ABCI applications can be leveraged to build increasingly complicated autonomous applications and to ensure that production-grade code can be reused in different contexts.

The composition of applications is further discussed in the Appendix.

d. On-Chain Protocol to Anchor Autonomous Services

The on-chain protocol anchors the Autonolas autonomous services – and in particular the current incarnation as agent services – on the target settlement layer and provides the primitives needed to create, operate and secure such services. Autonolas benefits from a modular design with a clear separation of concerns and opportunity for extensibility without compromising its security and permissionless nature.

From an architectural point of view, the contracts satisfy the following properties:

- Follow a core-periphery architecture (such as in Uniswap), which allows for changing out periphery functionality without changing the data models at the core.

- Allow for extension via modules (such as in MakerDAO).

The protocol is not directly upgradable and instead relies on various upgrade alternatives. Most prominently, it features upgradable modules, strategies, and parameters, among others. Direct protocol upgrades are achieved by launching a new version of the protocol while keeping the old deployment intact. Developers using the Autonolas agent protocol will need to actively migrate to newer versions of the protocol.

Examples of modules include governance and staking. Governance is particularly important in a modular system, as it is used to vote on the adoption or abandoning of modules. By ensuring an immutable core, the Autonolas protocol provides guarantees that once created, the ecosystem's agent components, canonical agents, and services NFTs are not mutable by governance – an important guarantee of censorship resistance.

The Autonolas on-chain protocol is built with the open-aea framework in mind as the primary framework for realizing agent services. However, it does not prescribe the usage of the open-aea and allows for services to be implemented on alternative frameworks.

i. Elements

Agent Component

A piece of code together with a corresponding configuration that is composable into agent code. In the context of the open-aea framework, it is either a Skill, Connection, Protocol, or Contract. Each component exists as code off-chain and is uniquely represented by an ERC-721 NFT with a reference to the IPFS hash of the metadata, which then references the underlying code via a further IPFS hash.[11] We describe on-chain code referencing via hashes in more detail in the Appendix (Section c). Sometimes, when it is clear from the context, we simply denote agent components as components without stating this explicitly.

Canonical Agent

A configuration and optionally code that defines the software agent. In the context of the open-aea framework, this is specifically the agent config file which references various agent components. The agent code and configuration are identified by its IPFS hash. As with agent components, each canonical agent exists as code off-chain and is uniquely represented by an ERC-721 NFT on-chain with a reference to the IPFS hash of the metadata, which then references the underlying code via a further IPFS hash.

[11] Although currently code is referenced using IPFS this nested design allows for forward-compatibility with other platforms like Arweave.

When it is clear from the context and not otherwise stated, we simply denote canonical agents as agents without stating this explicitly.

Whether the author of either a component or an agent correctly makes this association is up to them (i.e. we follow an optimistic design approach). Autonolas tokenomics – discussed below – incentivizes the correct mapping.

Agent Instance

An instance of a *canonical agent*, running off-chain on some machine. This runs the agent code with the specified configuration. At a minimum, each agent must have a single cryptographic key pair whose public address identifies the agent.

If not clearly stated, for instance when there are mentions of operators, agent instances are simply denoted as agents without stating this explicitly.

Service

A service is made up of the following:

- A set of canonical agents
- A set of service extension contracts (defined below)
- A number of instance agents per canonical agent
- A number of operator slots

A service defines the block time at which it needs to fill slots and when agent instances need to go live. Each service is an ERC-721 NFT.

Service Multisig or Threshold-Sig

A multisig associated with a given service. It contains the assets the service owns and validates the multi-signed transactions arriving from the agent instances. The protocol supports multiple multisig implementations that the service owner can choose. Most prominently, Autonolas builds on Gnosis Safe, the popular and well-audited multisig for which the protocol calls the Gnosis Proxy Factory to create a Proxy instance.[12]

Service Extension Contract

A set of custom on-chain contracts that extend the functionality of a service beyond what the agent protocol offers. These contracts are outside the scope of the Autonolas on-chain protocol. The service extension contracts are also responsible for defining the service-specific payment plan under which operators and the Autonolas protocol (and therefore developers) are rewarded.

[12] The Proxy does not contain any substantial logic. For signature validation and other utilities, it calls into the Proxy Master, a singleton contract.

Third-Party Contract

Any contract which might be called by the service extension contract or by the protocol. These contracts are outside the scope of the Autonolas on-chain protocol.

ii. Roles

Developer

Develops agent components and canonical agents. Registers agents and components. Registering involves minting an NFT with reference to the IPFS hash of the metadata which references the underlying code.

Service Owner

Individual or entity[13] which creates and controls a service. The service owner is responsible for all aspects of coordination around the service, including paying operators and ensuring the agent code making up the services is present and executable.

Operator

An individual or entity operating one or multiple agent instances. The operator must have, at a minimum, a single cryptographic key pair whose address identifies the operator. This key pair must be different from any key pair used by the agent. Operators can register for a slot in a service by using an agent instance address they control as well as their operator address.

Service User

Any individual or entity using a given service. These users are outside the scope of the Autonolas stack. It is the responsibility of a service owner to incentivize users to use their service.

iii. Core Smart Contracts

Core smart contracts are permissionless. Autonolas governance controls the process of service management functionalities and of minting new NFTs representing components and agents (i.e. it can change the minting rules and pause minting). The remaining functionalities, in particular transfer functionalities, are not pausable by governance.

[13] This includes institutions, companies, machines, etc.

GenericRegistry

An abstract smart contract for the generic registry template which inherits the Solmate ERC-721 implementation.

UnitRegisty

An abstract smart contract for generic agents/components template which inherits the GenericRegistry

Component Registry

An ERC-721 contract that inherits the UnitRegistry and represents agent components.

Agent Registry

An ERC-721 contract that inherits the UnitRegistry and represents agents.

Service Registry

An ERC-721 contract that inherits the GenericRegistry, is used to represent services and provides service management utility methods.

Autonolas extends the ERC-721 standard to support appending additional hashes to the NFT over time. This allows developers and service owners to record version changes in their code or configuration, and to signal it on-chain without breaking backward compatibility.

iv. Periphery Smart Contracts

Periphery contracts are fully controlled by governance and can be replaced to enable new functionality, but also to restrict existing functionality.

GenericManager

An abstract smart contract for the generic registry manager template.

Registries Manager

A contract inheriting from GenericManager via which developers can mint components and agent NFTs.

Service Manager

A contract inheriting from GenericManager via which service owners can create and manage their services.

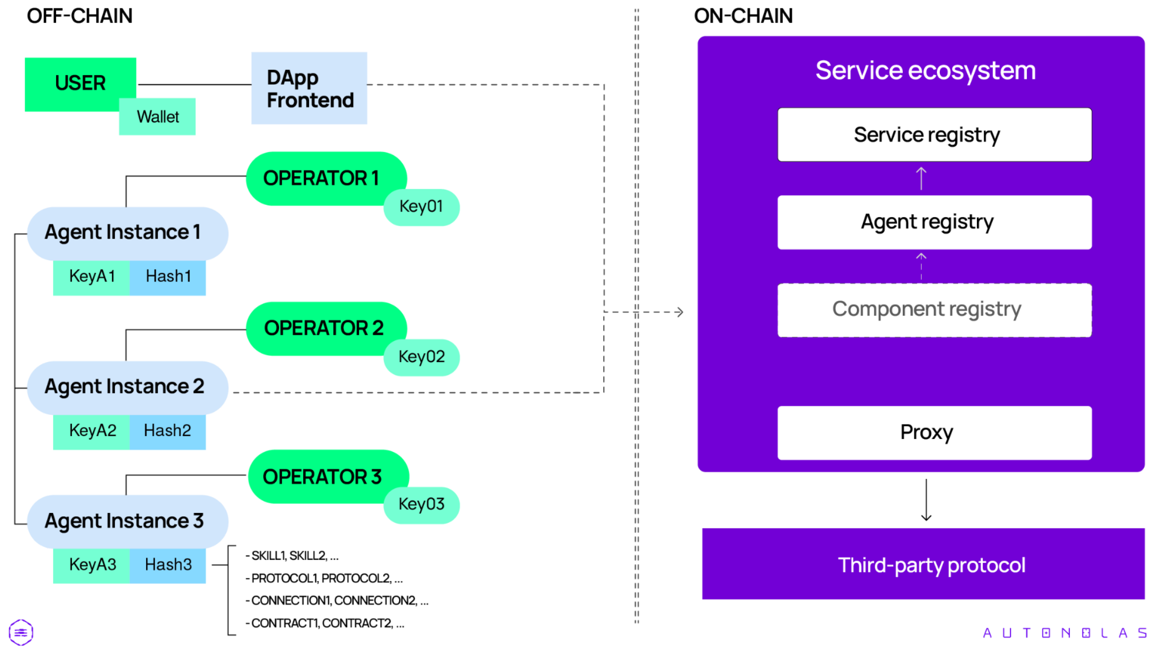

Fig. 4: Basic Autonolas Services Architecture.

v. On-Chain Protocol Workflow

Next, we give a succinct description of the basic workflow of the on-chain protocol.

Generally speaking, the standard workflow starts with a developer building either an Agent Component or a Canonical Agent. Alternatively, a given Service Owner starts the workflow by publishing a specification of an Agent Component or Canonical Agent and then asking developers to provide an implementation for them. Next, the developer uses the corresponding on-chain – Component or Agent – registry to register a representation of the new code on-chain in the form of an NFT. Developers have the responsibility to mint the NFTs representing their code at their own expense.

During registration, the following information must be included:

- Unit type of the code which can be Component or Agent
- Address of the owner of the code (usually the same as the developer)[14]
- Component dependencies in sorted ascending order of the component or the agent that needs to be registered
- IPFS hash that references the agent component/canonical agent metadata.

[14] This address represents the agent/component owner and it is the one eventually rewarded by the tokenomics.

Through minting, the specified owner becomes the holder of the NFT representing the agent component/canonical agent. Off-chain, the metadata file matching the IPFS hash of the registered agent component/canonical agent metadata is pinned on an IPFS node for later retrieval. The metadata file contains a reference to the component name, its description, a `code_uri` pointing to the code making up the component, and an `image` URI which is used to visually represent the NFT. Together, the on-chain protocol and the off-chain code storage enable code sharing and reuse by providing a place to contribute canonical agents or agent components.

Next, a Service Owner creates a Service from Canonical Agents and registers the corresponding Service in the Service Registry. The information that needs to be added to the registry includes the following:

- Address of the owner of the Service
- NFTs that represent the Canonical Agents that make up the Service
- Number of Agent Instances to be created per Canonical Agent and required bond to register an Agent instance in the service
- Threshold or minimum number of Agent Instances that can sign the multisig created for the Service
- IPFS hash that references the service metadata.

Similar to agents and components, the metadata file contains a reference to the component name, its description, a `code_uri` pointing to the service configuration, and an `image` URI which is used to visually represent the NFT.

Once the Agent Operators have been registered and the actual Agent Instances for a given Service are filled, the Service Owner instantiates a multisig with all Agent Instance addresses configured and the threshold stipulated (see Fig. 5).

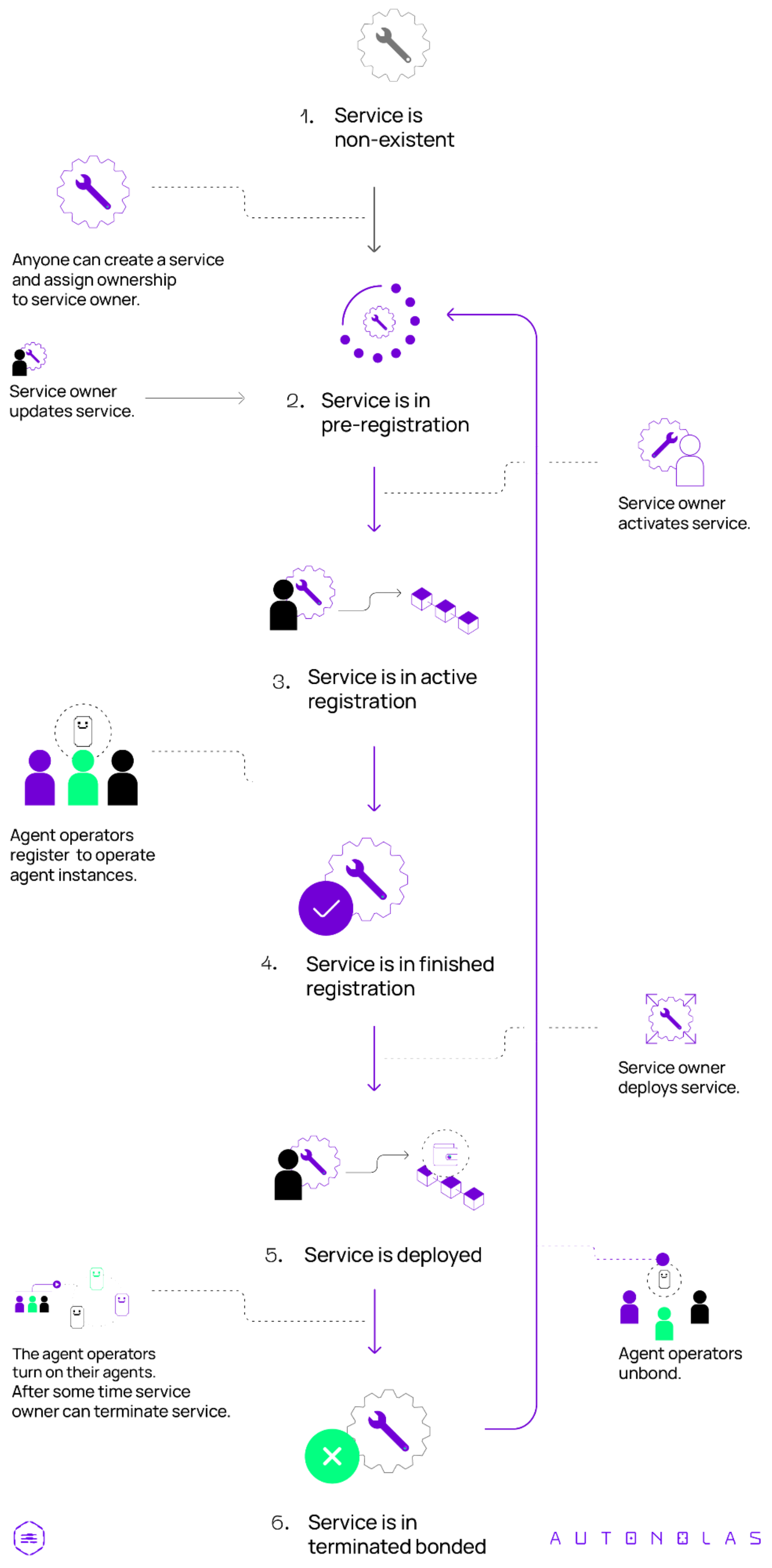
1. Service is non-existent
Anyone can create a service and assign ownership to service owner.
Service owner updates service.
2. Service is in pre-registration
Service owner activates service.
3. Service is in active registration
Agent operators register to operate agent instances.
4. Service is in finished registration
Service owner deploys service.
5. Service is deployed
The agent operators turn on their agents. After some time service owner can terminate service.
Agent operators unbond.
6. Service is in terminated bonded
AUTONOLAS

Fig. 5: Lifecycle of an Agent Service.

Once deployed, the service multisig is by default under the full control of the agent instances, and therefore under the operators' control. To ensure the economic safety of the service, the service owner can require a slashable deposit from the agent operators. Furthermore, a final defense against operator malfeasance can be added via Gnosis Safe Modules. This involves a manual override of the service (always acting via the multisig) by temporarily disabling the agents' ability to control the multisig.

vi. L2 deployments and efficiency

Gas efficiency

So far, the design assumes a monolithic deployment of the on-chain protocol on a single blockchain, likely Ethereum.

The current design is not particularly gas-efficient. The dependency structure between agents and components, and between components and their constituent (sub-)components can be complex and result in a large dependency tree. Updating a single hash deep down in the dependency tree technically requires updating all hashes. This is a gas-intensive and somewhat unnecessary operation. Indeed, the top of the tree – e.g. the service – resolves off-chain the entire tree by design.

In a more gas-efficient design, only the service has the most up-to-date hash, and a component just maps its dependencies once, never updating its hash henceforth. Since the service resolves the agent, and the agent resolves, in turn, the component, we have integrity off-chain anyway. This design however presents a downside. Namely, consuming components and agents directly from the on-chain registry is no longer feasible, as the hashes present there are likely out-of-date.

L2 deployments

In order to cater to a multi-chain world, it should be possible to register agent services on arbitrary chains. To avoid recreating a full copy of the Autonolas protocol on each chain, simplified versions of the Service Registry and Manager contracts can be deployed in any new chain that do not require the presence of the Agent and Component Registries. As it has been discussed above, from a functional perspective, Component and Agent representations on-chain largely serve discovery. In the section Tokenomics, we also introduce their economic relevance. It is sufficient to represent Components and Agents on a single chain – where the full protocol is deployed – at a gas-opportune time. Services on L2s can then reference these Agents and Components optimistically.

vi. Threat Model for Agent Services

Autonomous services work under the following threat model:

- A service is managed by a service owner, who is in charge of managing the service lifecycle (e.g. sets it up and can shut it down, see Fig. 5)
- A service is run by a set of operators, each running at least one agent instance, for a total of *n* agent instances
- Every pair of agent instances in the service can securely and independently communicate
- A majority of the *n* agent instances run the agent code defined by the service (typically at most ⅓ of the instances are allowed to be malicious for the service to be guaranteed to run)
- A malicious agent instance can deviate arbitrarily from the code that is supposed to run
- A service is registered in a L1/L2 blockchain from which the economic security of the service is bootstrapped
- Every operator must lock a deposit for each agent instance they own in the L1/L2 blockchain where the service was registered
- Agents can punish each other's misbehavior by submitting fraud proofs to each other at the service level, causing – if accepted by the other agents – slashing of the deposit of the malicious instance on the target chain
- The service owner locks a deposit equal to the total deposits requested from the agent instances. This is used to incentivize the service owner to release the agents deposits at the end of the lifetime of the service

An autonomous (or agent) service is decentralized by virtue of minimizing the trust placed on individual agent instances. Service owners are assumed to act honestly by default. This is a natural assumption for when the service is not yet live, as the service owner is the only one defining and controlling the service at this point. Once the service is live the service (specifically its multisig) is under the control of the agents. Once the service owner decides to terminate the service, the control over the service multisig returns to the service owner. In cases where the service owner wants to reduce their control over the assets controlled by a service multisig, they can introduce further smart contract constraints to limit their control (e.g. either by placing the assets into escrow or by turning the service owner itself into a proxy contract with limited functionalities, e.g. limited to only interact with the service manager and unable to directly interact with the service multisig). We consider this outside the scope of Autonolas protocol.

vii. Incentive Compatibility for Agent Services

Since malicious actors are potentially present at each level of a decentralized application, shared security and shared trust are crucial. It is thus essential to build robust and secure off-chain agent services. Agent services, by their nature, can rely on various sources of trustworthiness.

First of all, we have seen that agent services are powered by the SCG which allows them to leverage a *non-adversarial majority* source of trustworthiness.

However, agent services can leverage an additional source of trustworthiness defined as having *high adversarial cost* via an **incentive compatibility module**. The latter is a mechanism that allows *detecting* and *punishing* misbehavior as well as *rewarding* honest reporting of misbehavior in such a way that adversarial acts give smaller benefits than acting honestly.

Recall that an agent service can be described as a set of agents working together for a unique goal. The incentive compatibility module is an extension component that allows increasing the predetermined goals of the agents running the service itself.

More precisely, in addition to the typical actions required to perform the service with this module, agents can detect malicious or incorrect activities performed by one or more actors belonging to the same service. As such, some or all of the agents in the service also play the role of monitoring and reporting any rule-breaking committed by one or more other agents.

If one of the agents considers a certain action Byzantine it can raise a blame claim against the actor of the action. Such a claim can be used to trace the reputation of agents and their operators.[15]

If a majority of the agents have agreed on the malicious action of an agent, the latter can be punished by slashing a portion of the capital locked by its operator. Moreover, an agent is incentivized via rewards to raise honest alerts. As a result, the incentive compatibility module ensures that more is gained by acting within the rules of the protocol than by acting against them.

To summarize, each agent's service can be equipped with internal detection of misbehavior. By means of slashing, taking action against the service is disincentivized, and by means of reward, falsely reporting misbehavior is disadvantageous. The extent to which the incentive compatibility module is reusable across services is a pending question.

It is worth noting that an agent service can also leverage a more classical way for misbehavior detection through ad-hoc actors whose aim is that of controlling the behavior of the service. For example, similar to what ChainLink proposed with the AntiFraudNetwork [13], it is possible to develop an agent's service whose unique goal is that of controlling the behavior of the actors making up a specific service and

[15] For example, as suggested by [12], it is possible to represent the reputation of an operator with an NFT, which is unique as a single proof among the operators, enabling reputation evaluation universally.

triggering penalties in case of unauthorized activity. The limitations of this approach, however, are that external services are only able to report on externally visible behaviors of other services, and are unable to detect and punish internal misbehavior.

3. Tokenomics

Autonolas project's scarcest resource is agent developers. Autonolas' tokenomics gives due consideration to attracting and retaining developers, as well as facilitating the composability of their code contributions, to enable code to become more than the sum of its parts. Additionally, the tokenomics are geared towards attracting bonders, who can make their capital useful, by pairing it with code. Finally, we introduce a novel model of production which enables the protocol to own its own gainful services.

a. Overview

i. Key Goals

Autonolas tokenomics pursues three goals:

- **Pairing Capital and Code**
  The first goal of Autonolas tokenomics is enabling the pairing of capital and code in a permissionless manner. Autonolas operates on the following assumption: code on its own does not communicate its value, and capital on its own is not productive. Combined, these tenets reinforce each other. By pairing quality code with capital, Autonolas addresses a key problem faced in open-source development: developer time is a scarce resource subject to insufficient remuneration.

  Autonolas leverages a bonding mechanism (see Level 1: Bonding) to grow the protocol's capital in the form of Protocol-owned Liquidity. It uses a novel *staking* model for code to allow developers to track their code contributions on-chain (see Core Smart Contracts) and receive rewards for its *usefulness* (see Developer Perspective and Measuring Contribution).

- **Enabling Protocol-owned Services (PoSe)**
  The second goal is to create a flywheel that attracts increasingly more value and provides truly decentralized *protocol-owned* (autonomous) services, owned by a DAO, operated by the ecosystem, and implemented by developers around the world. This novel primitive enables the DAO that manages the PoSe to own productive autonomous services and derive donations from them. Notably, Protocol-owned Services provide the level of (off-chain)

decentralization necessary for the longevity of the ecosystem in the face of uncertain regulation.

Autonolas aims to accomplish this goal by acquiring and productively deploying *donations* in the ecosystem (see. Ecosystem Flywheel). Such *donations* may arise from Autonolas PoSes, e.g. PoSes managed by the Autonolas DAO, and third-party PoSes, e.g. PoSes managed by third-party DAOs (see Protocol Business Model).

- **Incentivising Composability**
  The third objective of Autonolas tokenomics is to enable and incentivize software composability. In a nutshell, "a platform is composable if its existing resources can be used as building blocks and programmed into higher order applications."[16] Composability of software components in an ecosystem enables exponential returns. The on-chain protocol enables this composability allowing NFTs which represent code and services to be registered and be combined together into higher order applications (see Core Smart Contracts) for the protocol and across DeFi.

  The composability is then extended by the tokenomics that, as mentioned above, allows developers to track their code contributions on-chain (see Core Smart Contracts) and receive rewards for its *usefulness* (see Developer Perspective and Measuring Contribution).

ii. OLAS Token

The protocol coordinates the goals described above through a tradable utility token, OLAS, that will provide the access to the core functionalities of the Autonolas project. The token follows the ERC-20 standard and will be deployed on the Ethereum mainnet.

The token has an inflationary token model to account for the economic primitives enabled by Autonolas tokenomics, e.g., bonding mechanism (see Level 1: Bonding) and OLAS top-up (see Developer perspective).

Functionalities enabled by OLAS

We summarize the main functionalities enabled by OLAS as follows:

1. OLAS can be locked for veOLAS to participate in the Autonolas DAO governance, thus shaping the protocol and its tokenomics (see Role of Governance and Level 3: off-chain signaling).

[16] Jesse Walden, *4 eras of blockchain computing: degrees of composability* [14]

2. OLAS can be locked for veOLAS for permissionless access of a service whitelist that unlocks code owners' top-ups (see Service-Owner Perspective).
3. OLAS can be used to acquire (on a third-party DEX) LP-tokens that are required for the bonding mechanism (see Level 1: Bonding). This will enable protocol-owned-liquidity and therefore support the protocol's long-term growth.

OLAS inflation model

The number of OLAS tokens is capped at 1bn for the first 10 years and the maximum token inflation per annum is capped at 2% thereafter. An allocation of

- 32.65% of the tokens were distributed to founding members of the DAO;
- 10% of the tokens were distributed to Valory AG to maintain, run and further the Autonolas protocol;
- 10% of the tokens were allocated to the Autonolas DAO treasury; and
- 47.35% may be used to incentivize developers' top-ups for useful code and bonders, autonomously provisioned by the protocol over the initial 10 years.

The protocol aims for an s-shaped curve of token emissions (see Fig. 6).

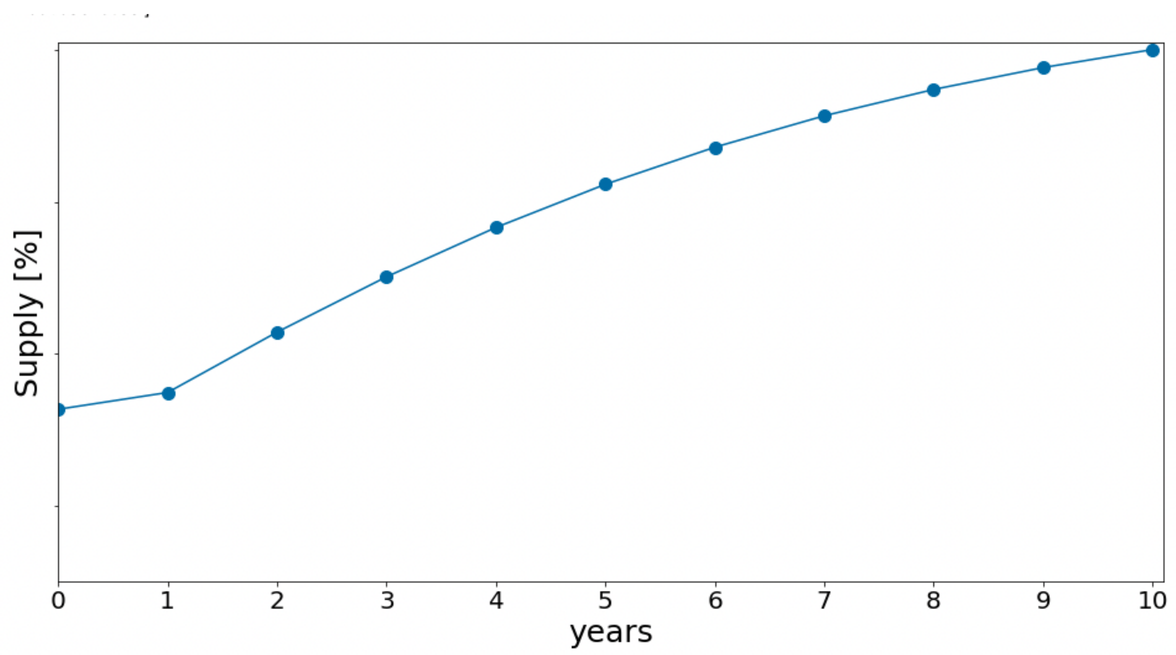

Fig. 6: In the first 10 years a maximum of 47.35% of the token supply could be issued to the ecosystem via component and agent top-ups incentives, and bonding. A low (up to 2% p.a.) level of token inflation can be maintained thereafter.

iii. Protocol Business Model

Autonolas protocol aims to facilitate rewarding capital and code contributions from newly-minted OLAS (see Ecosystem Flywheel, Level 1: Bonding, Measuring Contribution). But where do we expect the majority of protocol profits (i.e. profits accruing to the treasury) to fund other ecosystem initiatives (e.g. grants) to come from?

Autonolas considers three primary contributors:

- **Autonolas Protocol-owned Services**: The protocol can use governance to run services whose profits are then returned to the protocol in form of donations. Part of such donations (those not returned to useful component/agent developers) are retained by the treasury.
- **Protocol-owned Liquidity**: This can either be deployed to third-party protocols for purposes like providing liquidity in decentralized exchanges, or put to work in Protocol-owned Services. All profits returned from PoL are under the treasury's ownership.
- **Third-party services (incl. third-party PoSe):** A limited amount of the protocol's profits will likely come from the treasury's share of the voluntary donations from third-party services.

Slashed agent operator bonds (see Agent Operator Perspective) are another, likely limited, source of treasury profits.

> ★ **Example of a Protocol-owned Service – Oracle**
> Protocol governance can launch an oracle service and act as the service owner. This means that the service is fully specified and managed via the governance process of the Autonolas DAO. For practical reasons, a subDAO of the governance may be responsible for curating the service. The service is then run by third-party operators, and future code improvements are delivered by third-party developers. Any profits accruing from the oracle's operation might be donated to the protocol and shared with code NFTs owners that facilitated the service (see Measuring Contribution).

Next, we detail how Autonolas tokenomics focuses on core roles in the project, namely bonders, lockers, service owners, and developers.

b. Levels of Tokenomics

Autonolas tokenomics model can be broken down into three levels. Each level has its own objective and target audience.

i. Level 1: Bonding

Summary: Holders of OLAS can provide liquidity in a Decentralized Exchange (DEX) and sell their corresponding liquidity provider (LP) tokens to the protocol for OLAS at a discount. The process is similar to bonding in Olympus DAO [15], albeit with some important differences.

Objective: This creates a continuous fundraising market for the protocol and ensures there is ongoing liquidity for the protocol token OLAS, thus contributing to price stability. Furthermore, it endows the protocol with assets it can put to productive use in Protocol-owned Services or any other yield-bearing investment opportunities.

Audience: OLAS holders that aim to contribute to the project by providing liquidity.

An OLAS holder with whitelisted LP assets (for example OLAS-DAI, OLAS-USDC or OLAS-ETH, from a single DEX[17]) can bond those assets via the bonding contract at time t. Then at time $t + t_v$, such token holder, i.e., the bonder receives OLAS at a discount, measured by a Discount Factor[18], relative to the price quoted on the relevant DEX at time t. For this mechanism, two variables must be present at time t:

- Bond Price = (Discount Factor) * (DEX price for OLAS)
- Time of Vesting t_v

The time of vesting t_v is set by governance as a parameter on the bonding contract. Based on the above, the bonder is quoted the OLAS they are entitled to at the end of vesting their bond. The LP assets bonded contribute to the Protocol-owned Liquidity. A diagram depicting OLAS bonding mechanics can be found in Fig. 7.

[17] Such as SushiSwap or Uniswap, for example.

[18] The Discount Factor is constantly fluctuating and it is discussed later, see Discount Factor and Bonding.

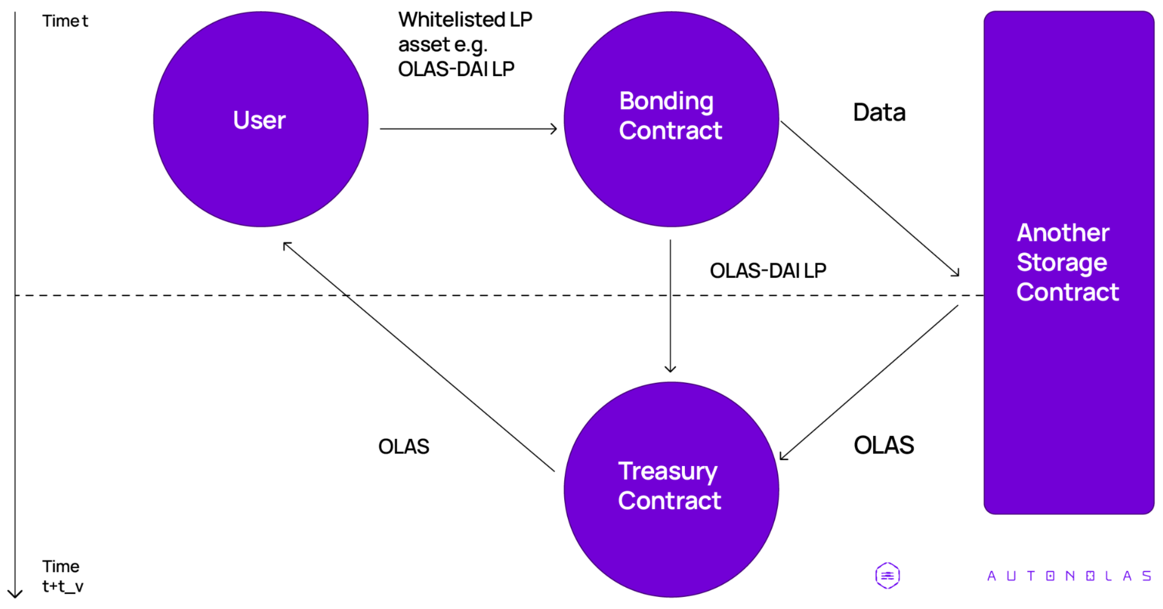

Fig. 7: Bonding OLAS.

What we have presented up to this point is identical to Olympus DAO v1 [15]. Where our tokenomics differ is in innovations on the concept of bonding. Namely, a control mechanism is introduced to ensure that the growth of Protocol-owned Liquidity ("Capital") is proportional to the growth of the components and agents ("Code")[19] usefulness[20], e.g., Code productively deployed in autonomous services.

Such a control mechanism is, in the first instance, enforced by several built-in on-chain metrics which are used to estimate the potential for the production of code through the usage of a *production function,* PF. The details of the production function are outside the scope of this document. However, it is worth noting that it will be possible to change the production function model's parameters via governance.

The bonding model and component/agent top-ups result in an inflationary token model, as new OLAS need to be minted. Notably, an inflation schedule (see OLAS inflation model) is used to limit how much the protocol can mint per epoch. In turn, since bonding implies minting new OLAS tokens, there is a maximum amount that can be bonded at any given epoch.

[19] Here, the terms *components* and *agents* are to be understood as *agent components* and *canonical agents* respectively as defined in the Section Elements.

[20] The usefulness of Code is intended in terms of total donations it facilitates to the Protocol. This will be autonomously determined by the Autonolas Protocol via built-in on-chain metrics.

Discount Factor and Bonding

To control the amount of capital that can be bonded, the protocol will leverage a control mechanism via a production function *PF.* In the initial implementation the mechanism looks as follows: the larger the current value of PF is, the greater the discount factor at which bonders will receive OLAS, and the smaller PF, the smaller the discount factor, while respecting the inflation schedule.

This control mechanism enables "push" and "pull" dynamics when there is a high potential output in the production of useful code that, in turn, could be productively deployed in autonomous services. The "push" results from an advantageous discount that a bonder can receive in purchasing bonds. The "pull" results in the willingness of the OLAS holder to provide liquidity to the protocol thus contributing to token price stability.

This way the protocol can autonomously guarantee that bonding is proportional to the useful code in the system. This implies there will be periods when it will not be possible to bond to the protocol – e.g. when capital available in the protocol exceeds code – and the profitability of bonding will vary over time.

The bonding is not directly reversible[21]. That is, there is no direct mechanism for the protocol to buy back OLAS. One expects the value of OLAS to reflect the usefulness of the code in the ecosystem, the capital bonded to the system, as well as the usefulness of the services owned by the protocol.

The bonding is only available to holders of OLAS from a secondary source (e.g. a DEX) as OLAS is needed to get LP tokens.

ii. Level 2: Locking

Summary: Holders of OLAS lock their OLAS in return for a *non-tradeable* virtualized claim called veOLAS. Specifically, veOLAS is a virtualized, non-tradable and non-delegable governance right towards the Autonolas Protocol. Only holders of veOLAS can participate in the governance of the Autonolas DAO.

Objective: This aims to create alignment between token holders that lock OLAS and the protocol objectives. It incentivizes committing to the token OLAS by locking it to obtain veOLAS and being able to steer and protect the protocol via governance. It reduces short-term decision-making in governance vis-à-vis a model with a tradeable governance token.

[21] Olympus DAO offers "inverse bonds" to allow OHM holders to sell OHM at a premium to the protocol in exchange for protocol-owned assets. That makes sense because OHM is an asset that is meant always to be priced above risk-free value, thus requiring an inverse mechanism. Unlike in Olympus DAO, the risk-free value of the Autonolas protocol is not easily measurable, and OLAS does not attempt to be a stable asset a priori.

Audience: Holders of OLAS.

Holders of OLAS lock their tokens and receive a virtual representation called veOLAS in return (see Fig. 8). The virtualized veOLAS is a non-tradable and non-delegable voting right that gives the user a claim on locked OLAS against the locking contract. The user must specify the locking duration in weeks, up to a maximum of 4 years. The locking length determines the multiplier which boosts voting rights. Voting rights decrease linearly over the locking duration (like in Curve).

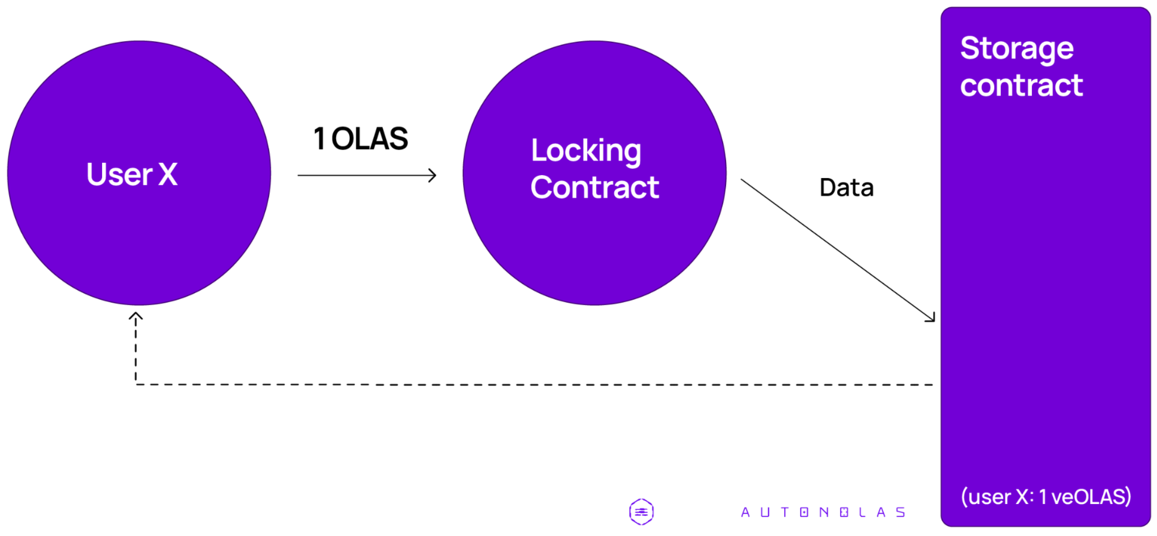

Fig. 8: Locking OLAS for veOLAS.

Holders of veOLAS can participate in on-chain governance of the protocol as well as, eventually, any off-chain signaling (see Level 3: Off-chain Signaling). All holders of veOLAS are collectively the voting members of the Autonolas DAO (see Governance).

Notably, holders of veOLAS control governance parameters that determine how donations accrued by the Protocol and newly minted OLAS are shared across two recipients (Code owners of NFTs representing Code and protocol treasury) of payouts in Autonolas tokenomics (see Fig. 9). Holders of veOLAS are incentivized to maximize the utility of the OLAS token which they have locked away in return for governance rights. See Role of Governance, Service-Owner Perspective, Level 3: Off-chain Signaling in order to identify the various action points of veOLAS holders.

When the lock phase has passed, the veOLAS holder can retrieve their locked OLAS. Locking can also be renewed, including before the current lock phase has ended.

The locking mechanism creates “push” and “pull” dynamics. In the bootstrapping phase of the protocol, "push" results from the potential governance impact on adjusting yields and reward boosts for early code NFTs owners. After bootstrapping, the “push” also results in the ability to signal valuable Code (see Level 3: Off-chain Signaling) that can be productively deployed by veOLAS holders in Autonolas-owned autonomous services. The “pull” results from OLAS holders being incentivized to participate in governance to increase the demand and the utility of OLAS.

iii. Level 3: Off-chain Signaling

★ **Note**: Level 3 will not be active in the early days of the protocol’s operation.

Summary: veOLAS holders can participate in protocol governance off-chain and form subDAOs to enrich on-chain contribution metrics.

Objective: Level 3 decision-making allows holders of veOLAS to augment on-chain built-in measures with indirect contribution measures (ICM) constructed off-chain.

Audience: Holders of veOLAS with a particular interest in shaping the protocol.

Next, we share some ideas on possible Level 3 applications. An off-chain signaling game can be created to further enrich the metrics used by the protocol to calculate rewards for useful Code. Indirect contribution measures can be created off-chain at Level 3 and fed back into the protocol via governance or optimistically via an oracle mechanism to the Code reward calculations done on-chain.

Codes could be marked as legitimate according to some heuristics (e.g. being referenced in an Autonolas protocol-owned service) therefore lending the opportunity for an enriched incentive scheme. Off-chain signaling (i.e. single-use voting) could be used to allow veOLAS holders to signal the positive/negative contributions of Code. Notably, ICMs could express negative signals about components, agents, as well as services, and are thereby a useful element of an incentive-compatible design in the long run.

The exact implementation details of Level 3 can be left unaddressed, to be designed at a later date by the Autonolas DAO. The most current concern is to establish interfaces enabling off-chain decision-making to modulate on-chain measures via ICMs.

c. Ecosystem Flywheel

Autonolas provides a token-mediated coordination mechanism for the creation and operation of autonomous services for the four roles of service owner, developer, agent operator, and bonder. As such, it is a permissionless developer platform and has no responsibility or control over the services whose coordination it facilitates.

Many parts of the protocol can be permissionlessly used by anyone except some key functionalities which require holding veOLAS[22]. This reduces friction for ecosystem participants.

Next, we discuss the primary roles that make up the ecosystem, how they interact and when they benefit from holding the token.

i. Developer Perspective

The core problem open-source agent developers face in the absence of Autonolas is a lack of ability to monetize their own coding work outside traditional forms of employment or freelancing (each with its own drawbacks).

Furthermore, evaluating the value of agent/component code is not trivial. Typically, this evaluation is done by a central party. Autonolas seeks to avoid centralized evaluation, which by definition stifles permissionless ecosystem growth. Initially, to evaluate the code, the protocol can take advantage of several built-in on-chain metrics, most straightforwardly:

- Number of autonomous services a component/agent is used in that make some donations to the protocol
- Donations returned to the protocol from autonomous services

These values will be used to assign each component and agent their *direct contribution measure* (DCM) to protocol donations (see Measuring Contributions). That is, DCM measures the total donations that an agent or a component facilitates by being referenced in autonomous services whose owners provide donations.

Additionally, the ecosystem needs to attract capital to address the developer constraints and grow. The approach to solving these problems is to create a Code value market.

Autonolas protocol tokenomics seeks to incentivize talented developers to build Code that:

- Is *useful* for the protocol (e.g it attracts donations from services)

We see two paths in which value can be created permissionlessly.

Path 1

The developer develops a component or an agent (that exists as code off-chain), then mints its associated NFT (uniquely representing the component or the agent code on-chain). As the component or the agent is referenced in autonomous services that

[22] Specifically, veOLAS holdings are required for participating in the protocol governance (see Role of Governance) and for whitelisting autonomous services (see Service-Owner Perspective).

provide donations to the Protocol, its DCM appreciates over time. The more useful components/agents are built, the more donations the protocol might receive. So the higher will be the discount that a user receives in purchasing OLAS via bonding, thus bonders are incentivized to bond capital to the protocol.

Path 2

The service owner specifies a component or an agent, the developer develops said component/agent and mints the associated NFT, and the service owner returns donations to the protocol which compensates the developer.

The second path above will be deferred to a later protocol deployment stage because it is more intricate. The reason for this is that specifying a component is not trivial and leaves room for interpretation.

Furthermore, the service owner has an incentive to cheat the developer and reject a delivered component to match the specification, or to never return any donation from the resulting service to the protocol. However, this incentive of the service owner to cheat diminishes as their reputation comes into play. This is an interesting way to bootstrap new services, as we expect service owners will be incentivized to build up their reputation.

Therefore, returning to the first path, at a high level the Autonolas flywheel is designed to work as follows:

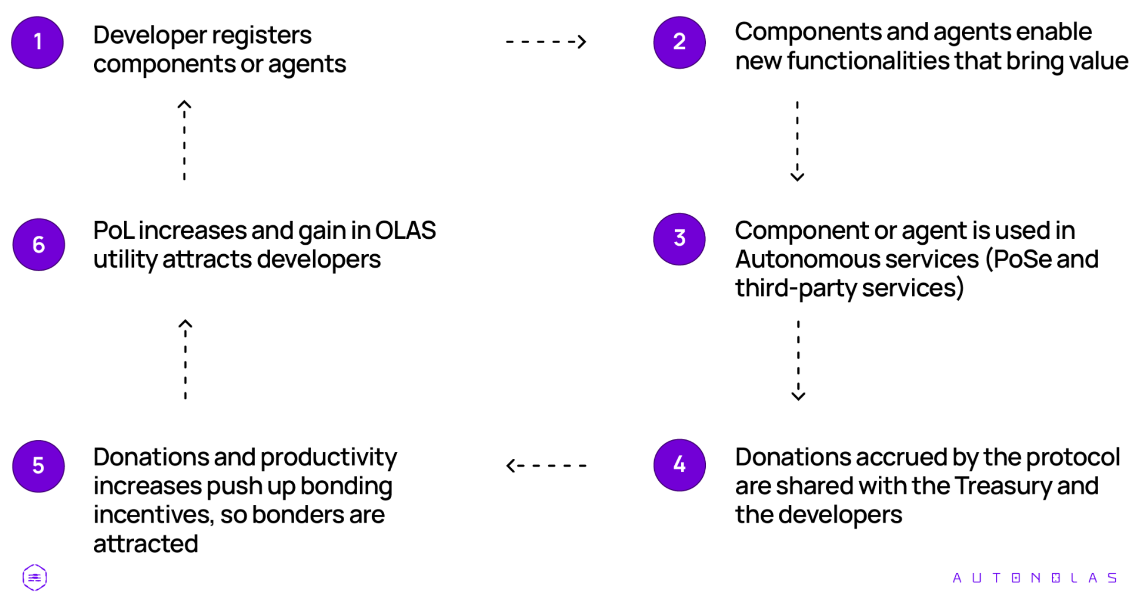
1
Developer registers components or agents
2
Components and agents enable new functionalities that bring value
3
Component or agent is used in Autonomous services (PoSe and third-party services)
4
Donations accrued by the protocol are shared with the Treasury and the developers
5
Donations and productivity increases push up bonding incentives, so bonders are attracted
6
PoL increases and gain in OLAS utility attracts developers
AUTONOLAS

Fig. 9: Ecosystem flywheel (Path 1).

1. The *developer* brings component or agent code and mints its associated NFT[23] uniquely representing the component or the agent code on-chain
2. The *component/agent* enables more functionality which adds value to the ecosystem
3. The protocol operates its own services made from agents and their components
4. The protocol rewards component/agent owners with donations generated from protocol-owned autonomous services, autonomous services owned by third-parties, and tops up[24] such rewards with the newly minted OLAS
5. The *bonder* brings capital (in the form of LP tokens) to (A) buy into the protocol and (B) gain a discount on the OLAS price. The discount increases the higher are the downstream protocol donations. Protocol-owned liquidity sees upwards pressure.
6. The deployment of Protocol-owned Liquidity and the increase of OLAS utility attracts developers to contribute with more Code

Notably, the above shows that developers can benefit from their contributions to the ecosystem by focusing exclusively on providing valuable code, and leaving to others the operation, management, and creation of services in which the code is deployed. Developers can opt to hold the components they create or sell them on an NFT marketplace (which results in the new owner receiving all future royalties for its use).

ii. Service-Owner Perspective

It is assumed that services that are not owned by the Autonolas protocol have their own tokenomics where applicable, and that they consume Autonolas as an infrastructure layer. In any case, DAOs and other entities acting as service owners of an autonomous service have an ongoing need to maintain and improve their service as well as to reward service code contributors. To this end, service owners can utilize Autonolas protocol as a fair reward distribution mechanism.

Specifically, service owners can proactively send funds, called *donations*, denominated in the blockchain's native currency (e.g. ETH) to the Autonolas protocol.

[23] In the case where one component/agent is developed by multiple developers they can create a multisig which owns the component NFT. Furthermore, minting can be subsidized by donors.

[24]If the agent or the component is referenced in a Autonomous service whose owner donates to the protocol and locks OLAS for veOLAS, then the Autonolas Protocol tops up the share of donations going to developers with freshly minted OLAS. The OLAS top-up is freshly minted OLAS acting like an "NFT staking reward": e.g. developers stake their "useful" code NFTs for whitelisted autonomous services and receive a share of the OLAS inflation in exchange.

Service donations are never made in OLAS. These are voluntary donations by the service owners. The protocol then proportionally allocates said donations to protocol treasury and code developers. The weight is determined by Autonolas governance. The share attributed to developers owning NFTs representing agents/components code is allocated based on the *direct contribution measures* (DCMs). The DCMs allow the Protocol to autonomously measure agents/components contributions in terms of the total donations they facilitate.

Hence some fraction of the donations accrued by the protocol can be paid out to the component/agent that facilitated them. The protocol can additionally opt to top up these royalties. Care has to be taken with top-ups as this could skew incentives to provide components/agents. This feature is therefore limited to whitelisted services. The whitelist is operated permissionlessly: service owners can opt into the whitelist by locking up OLAS for veOLAS to align long-term incentives. This has the benefit that OLAS is temporarily taken out of the market.

iii. Agent Operator Perspective

An autonomous service can consist of any number of agents. Every agent[25] is operated by an agent operator. The main role of these agent operators is to operate the autonomous service off-chain. Operators are not directly targeted with Autonolas tokenomics, as agent-services operation is a competitive business where we expect the marginal cost to be close to marginal revenue. It is up to the different services deployed on Autonolas to pay enough to attract agent-service operators. Hence, the protocol does not need to cater to operators directly.

Operators are paid by services, and the implementation of the compensation mechanism is service-specific. The funds for operating a service are provided by the service owner, either out of pocket or from the revenue of operating their service. The main expenditure of operators relates to running the agents, maintenance, and paying for gas expenditure incurred from settling transactions on-chain. Similar to node operators in a proof-of-stake blockchain such as Ethereum, agent operators can be requested by the service owner to provide a bond as collateral which is locked in a smart contract and might be slashed if the agent operator acts maliciously (in the sense that the agent controlled by the agent operator is malicious or faulty). This incentivizes honest behavior. If agent operators were not slashed the bond is refunded when the autonomous Service is decommissioned by the service owner and all agents have been deregistered. If agent operators were slashed, they get the non-slashed part back when the autonomous service is decommissioned by the service owner and

[25]Here, the term *agent* is to be understood as an *agent instance* as defined in the Section Elements.

all agents have been deregistered. The slashed part is henceforth owned by the Autonolas protocol.

In the medium term, we expect an operator market to emerge, potentially coordinated by a further smart contract layer.

iv. Bonder Perspective

Bonders have idle capital available and want to make this capital productive. Autonolas offers a solution to bonders that want exposure to OLAS and to become ecosystem participants. By providing liquidity in a Decentralized Exchange (DEX) and selling their corresponding liquidity provider (LP) tokens to the Protocol bonders receive OLAS at a discounted price in return for a lockup. The level of discount depends on several metrics, most notably on the amount of donations the protocol receives from service owners and by the number of developers that built useful code (that are measured by means of a product function PF, see Level 1: Bonding).

d. Further Elements of Tokenomics

i. Value Flow

Recall that the main inflows of capital for the Autonolas protocol are:

1. Profits from Autonolas Protocol-owned Services returned to the protocol via the service-donation endpoint
2. Donations from third-party (including third-party PoSe) services returned to the protocol via the service-donation endpoint
3. Profits from PoL returned directly to the treasury

veOLAS holders use governance to weight the proportionality in the allocation of inflows 1 and 2 to the various stakeholders: owners of agent/component NFTs and the treasury, respectively. These donations are denominated in the native currency of the blockchain where the protocol is deployed (e.g. ETH). Whitelisted services, i.e. autonomous services whose owners are also veOLAS holders, can direct further OLAS top-ups for owners of agent/component NFTs referenced in such services.

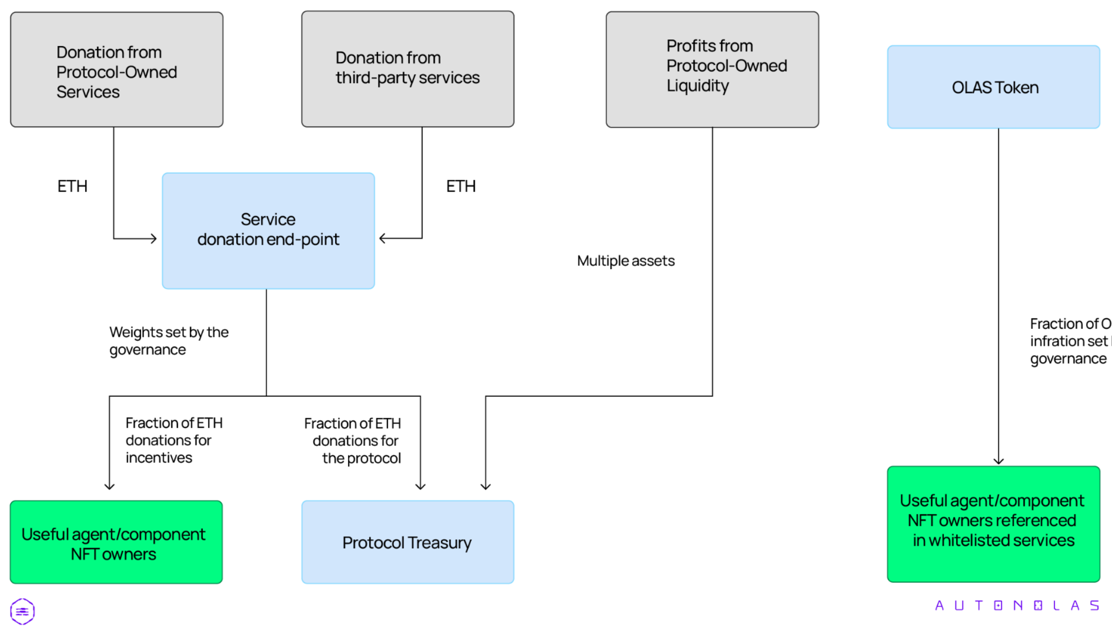
Donation from Protocol-Owned Services
Donation from third-party services
Profits from Protocol-Owned Liquidity
OLAS Token
ETH
ETH
Service donation end-point
Multiple assets
Weights set by the governance
Fraction of OLAS infration set by governance
Fraction of ETH donations for incentives
Fraction of ETH donations for the protocol
Useful agent/component NFT owners
Protocol Treasury
Useful agent/component NFT owners referenced in whitelisted services
AUTONOLAS

Fig. 10: Reward allocation from donations returned to the protocol. During the initial 10-year growth phase the rewards can be topped up with OLAS issuances.

iii. Ecosystem Bootstrapping

Autonolas Protocol-owned Services have been mentioned as one path for the Autonolas DAO to run services. In the bootstrapping phase of Autonolas, this approach can be extended to the DAO participants specing agent components, that after a successful governance vote, will be open for devs to implement and capture future associated component rewards.

iv. Measuring Contributions

To begin with, the protocol leverages the *useful code factor per agent/component* (UCFa/UCFc) metrics as *direct contribution measures* (DCMs) to autonomously quantify the utilization of agents and components in those services that return donations to the protocol. These measures detect the contribution of an agent or a component to the ecosystem in terms of the donations they facilitate by being referenced in services whose owners provide donations. As such, they are used to split the rewards to agent and component developers according to their contributions autonomously calculated by the protocol.

An epoch is a parameter measured in a fixed number of blocks and set by governance, and contribution measures are calculated based on this parameter. At each new epoch, the protocol allows for the recalculation of new DCMs and, in the future, the acceptance of new ICMs[26].

The useful code factor of a given component or agent (UCF_c/a) is the result of dividing the sum of all donations across all services featuring the component or the agents by the sum of all donations across all services. This implies a value between 0 and 1 for the UCF_c/a of a given component or agent.

v. Incentive Compatibility

The on-chain protocol has no knowledge of off-chain code. IPFS hashes referenced in NFTs are opaque from the perspective of the protocol. Furthermore, the protocol has no way of knowing whether dependency structures inherent to components, agents, and services NFTs are reflecting the actual dependency structure existing off-chain in code. Tokenomic design accounts for this by taking into account direct and governance-mediated indirect contribution measures. These are designed to create an incentive-compatible system for the on-chain representation of off-chain hosted code.

Below, we discuss a non-exhaustive list of attacks and how the protocol design mitigates them.

Spam minting

Nothing stops developers from minting multiple components or even copies of the same components. The mitigation leverages the fact that a minted component or agent incurs a creation cost, and that the tokenomics design renders component minting by itself worthless. Indeed, a component has a positive value in the protocol and sees a return only through being referenced in services that return donations to the protocol (i.e. showing up in a DCM) or being recognized via governance (in an ICM).

Copy minting

A hash of a given component can be reused across many components. Since the hash pointing to the code or configuration is itself embedded in a metadata file whose hash is referenced on-chain, there is no way to guarantee on-chain that hashes are referenced only once. The protocol optimistically assumes that in the presence of duplicate components, the service owners will coordinate to use the component registered by the original developer or the developer with the most activity. Ultimately, services rely on developers improving their components, and they have no interest in doing so if the rewards of their labor go to other individuals who have simply forked

[26] Indirect Contribution Measures were introduced in Level 3: Off-chain Signaling.

their code. Off-chain signaling can further help service owners discern which components are genuine.

Split minting

A developer can choose how to implement code in components. In the context of the open-aea framework, the intention is for the developer to implement one of four component types. However, the number of components in which a developer implements a given piece of functionality is entirely up to them.

Furthermore, a developer can reference empty components in their agents. As a result, there is potential for developers to inflate the number of components for a given functionality. The DCM & ICM need to collectively ensure that it is not profitable for a developer to cheat the protocol this way. Split minting implies that the protocol has no notion of a “base unit” for rewarding contributions.

Artificially inflating donations

It is, in general, not profitable for a bonder to inflate donations to increase the discount on OLAS purchased via bonding.

Incorrect dependency mappings

Dependency mappings in components, agents, and services cannot be validated on-chain. Mappings serve two primary purposes: allowing off-chain deployment tooling to build agents and services from on-chain specification, as well as ensuring the correct components are rewarded in DCMs and ICMs.

The protocol optimistically assumes that dependencies are correctly specified. Misspecifications in dependencies are disincentivized because of the following:

A. Service owners returning donations to the protocol have no incentive to reward components not used in and not useful for their service.

B. Service owners have an incentive to reward all components making up their service to ensure that developers contributing to the service are incentivized to continue doing so.

C. Developers are unlikely to see their components included in a service if they mis-specify their dependencies.

Off-chain reputation and ICMs can further contribute to ensuring that correct dependency mappings are created.

Incentives for reinforcing competitive behavior

Slashing of a security deposit of agents operators for violation of the service rules only gets us to discourage misbehavior. This however does not create direct incentives for operators running the corresponding agent code and/or outperforming other operators. Approaches to incentivize consistent high performance of operators may include: giving operators extra rewards for consistently hitting certain metrics, and factoring in operators' reputation in the system when calculating rewards.

vii. Cross-chain & Bridging

Services that operate on a given chain need Autonolas to be deployed on said chain. However, the tokenomic modules do not necessarily need to be deployed on every chain.

The first version of the full Autonolas protocol will be deployed on a single L1 chain – Ethereum. However, we envision Autonolas will feature rapid deployments of the service-related functionality on other major smart contract chains and prominent L2s. Since these deployments will initially be isolated from the main deployment on the tokenomics level, we refer to them as “island deployments”.

Even in this minimal cross-chain deployment, the agent services will be able to take action on multiple chains simultaneously. As Autonolas evolves toward full multi-chain support, it could take inspiration from MakerDAO and Gnosis’ Zodiac in the design of a cross-chain protocol v2 that also incorporates tokenomics. This would likely involve bridging DCMs between protocol instances in order to accurately measure service demand and code usefulness.

e. Emerging Properties

Autonolas’ tokenomics gives rise to a few properties informally described as follows.

- Each component and agent receives royalties as a share of the overall donations accrued by the protocol. Components’ and agents’ share of the protocol donations is determined by their usefulness – concretely, both the number of services they are used in and the usage of these services.[27] This means components and agents differ in the value they can create for their developers. This makes the mechanism that assigns value to software components incentive-compatible and acceptable to be permissionless.
- The protocol grows capital and useful code in tandem. Bonding is mediated via donations and developers that build useful components/agents. We expect the amount of both to grow proportionally.

[27] Usage of a service is measured by the donations the service returns to the protocol.

- Thanks to the Protocol-owned Liquidity, Autonolas owns capital which acts as a natural floor to protocol value and allows Autonolas to invest in productive services and the growth of the Autonolas ecosystem.
- Autonolas protocol design, in particular the usage of ERC-721 to represent components, agents, and services, should allow for the following:
    - Efficient composition with other DeFi primitives
    - Secondary market opportunities that enrich the functionality and usefulness of Autonolas.

We expect many useful extensions to emerge in the Autonolas ecosystem, including tools to facilitate the following:

- Collaboration between developers
- Coordination between developers and service owners to collaborate on the creation of new services
- Coordination of operators
- Aggregators for component, agent, service, operator, and developer reputation

Some of these might find inclusion in future protocol versions, and others can be realized by ecosystem members.

f. Tokenomics Smart Contract Architecture

A high-level description of Autonolas tokenomics smart contract architecture follows. This is subject to change and primarily serves illustrative purposes

Tokenomics

The main aim of this smart contract is to implement the logic of the mathematical model behind Autonolas tokenomics, such as the on-chain metrics which are used to estimate the potential for the production of code and the production function described in the previous section Level 1: Bonding.

Depository

This contract borrows concepts from OlympusDAO. It allows users to deposit LP-tokens (e.g. OLA-ETH) and creates a bond. When the bond vesting time is over, the user obtains a discount on the OLAS price. The depository contract borrows the logic to calculate the discount from the tokenomics contract.

Treasury

This contract also borrows concepts from OlympusDAO and contains the logic for the management of the Autonolas protocol Treasury. It allows the Depository contract to deposit LP-tokens assets in exchange for OLAS tokens. It also contains the logic to receive donations from Protocol-owned Services and to transfer tokens from the Treasury reserves. Finally, it has a management role in the Depository, Dispenser, and Tokenomic contracts.

Dispenser

This contract allows agent/component NFT owners to claim their incentives (e.g. the share of the donations autonomously assigned by the protocol to reward useful Code). It interacts with the Tokenomics contract to check the total rewards possible, from which rewards are calculated.

4. Use Cases for Autonomous Services

All decentralized applications that depend on either software or human-based off-chain processes are potential users of Autonolas services. We've identified two broad categories in which autonomous services can find early market adoption.

DAO Operations

Despite their early autonomous ambitions, most DAOs today rely on decidedly un-autonomous processes. Treasuries are suboptimally deployed by centralized parties. Protocol parameters are managed by overwhelmed and under-representative sub-communities, one painstaking vote at a time. DAO-to-DAO interaction takes place largely through Business Development teams. The high degree of centralization in DAO operations presents security, technical, liability and regulatory risks.

Protocol Infrastructure

Smart contract protocols still rely on suboptimal, opaque and/or third-party systems for running critical off-chain processes. For example, the derivatives exchange dYdX recently suffered outages because major parts of their stack were hosted on AWS, which was itself experiencing downtime [16].

What do Autonolas solutions look like in general for these categories? Below we cover concrete examples of how Autonolas autonomous services provide uniquely decentralized, ownable, scalable solutions.

⋆ **Note**: These categories are based on the current capabilities of the tech stack, our research to date, and the conversations we have had with projects operating in the Web3 space. We have also extrapolated out some future applications which, while less tangible at the time of writing, have strong future potential to thrive in the Autonolas ecosystem. Find these at Future Use Cases.

a. DAO Operations

As mentioned above, DAOs are currently very far from autonomous. Many mission-critical DAO processes are run by heavily centralized software deployed on traditional servers, by groups of trusted individuals (subDAOs) or third parties, and even decided by votes that permit all token holders (whole-DAO). Running DAOs this way introduces operational fragility, inefficiency, and centralization risk.

The two operations most problematic for DAOs are the following:

- Treasury Management
- Continuous Protocol Parameter Configuration

Below, we go into detail about what they currently look like from a process perspective, and some specific use cases. We describe a simple agent service for each use case and its benefits.

i. Treasury Management

Today, it is challenging for DAOs and multisig owners to achieve the full breadth of their treasury management goals. Organizations want to maximize the risk:return ratio of their on-chain treasuries, but are blocked by the functionality available to them. What's more, treasuries have grown to significant sizes, with over 60 sitting on more than $10m in assets, and over 15 with more than $100m [17]. On the multisig side, over $87bn dollars are currently custodied as ETH or ERC-20 assets in Gnosis Safe contracts [18].

In order to achieve their goals, DAO treasury managers have three main options:

- Vote in a strategy and have a subDAO execute on it
- Build out custom, single-use smart contract functionality to execute a certain set of transactions
- Automate treasury deployment through existing software practices

Voting in a strategy for a subDAO to execute poses several problems. First, subDAO participants need to be vetted, as they are capable of stealing funds. Second, those participants are humans who need to be recruited, managed, and compensated. Third, despite the resources expended by the DAO, these processes are not consumable or sellable to other DAOs. The list goes on.

Some DAOs have used tools like Aragon Agent [19] or custom smart contracts to directly vote in transactions to be executed on-chain. For example, Lido used Aragon Agent to deploy treasury assets to Unslashed Finance in order to secure insurance on a part of the assets Lido managed [20].

Technically, this achieves the DAO's goal in a crypto-native way. The transaction is executed robustly and the funds are never custodied trustfully. However, the process requires direct engineering resources to build and could conceivably suffer an exploit. Most critically, however, the code is used once and potentially never again. The initial opportunity, insurance cover, in Lido's case, expires and the DAO needs to retrieve the funds and re-vote to deploy again.

The third option available to DAOs is to run off-chain services using traditional third party cloud deployments. These examples are rarely publicized or documented because of the risks that they pose. If a project were to try to achieve what Lido did in the example above, namely purchasing insurance, they might write a bot to coordinate these purchases.

The advantage here would be that this operation could run continuously – it could be voted in once and run forever, purchasing insurance and securing assets indefinitely. However, the disadvantages and problems here are significant. The bot is to be fully trusted and vulnerable to downtime and censorship. It runs in isolation, and therefore cannot be audited or verified on an ongoing basis.

Furthermore, the operator of the bot exposes themselves to regulatory scrutiny and potentially disastrous liability implications. Therefore, software running in this closed-source environment sees little by way of the ongoing refinement it would see in an open-source environment. The result is that these services are far less robust and far riskier to use, and few projects entrust significant portions of their treasuries into their control.

Below, we briefly examine two use cases for autonomous services to significantly alleviate the aforementioned problems within the Treasury Management category:

- Yield
- Trading strategy execution, e.g. diversification & rebalancing

Yield

"This type of yield service is novel because it does not require that treasury managers constantly optimize between different opportunities, as is the case with existing on-chain yield aggregators."

As previously mentioned, significant value is now stored in on-chain treasuries. Our research shows that DAOs are eager to earn yield on their treasuries to boost their revenues. For the reasons described above, it is difficult for DAOs to earn yield in a manner that's highly secure and reliable, but also dynamic and efficient.

Autonolas yield services are a crypto-native way for decentralized treasury managers to achieve returns on their assets in DeFi.

An Autonolas service is a standard service as articulated in the On-Chain Protocol section (see On-Chain Protocol to Anchor Agent Services). It comprises components that integrate with various DeFi protocols – forming yield strategies – and with evaluative components which decide how to distribute assets across strategies to optimize overall performance.

An example of a strategy component is one that watches different liquidity pool positions on an AMM-based DEX, projects' Annual Percentage Yields (APYs), and shifts between pools to maximize yield. Higher-order evaluative components assess what proportion of capital should be allocated to this strategy versus, for example, a different strategy that optimizes yield across lending pools on a lending protocol.

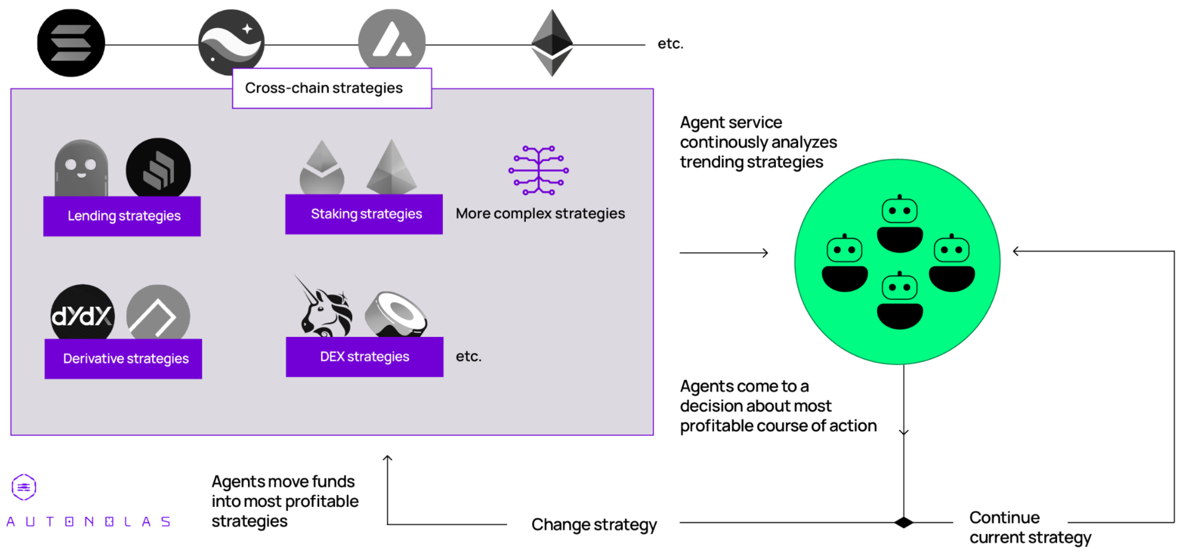
etc.
Cross-chain strategies
Lending strategies
Staking strategies
More complex strategies
Derivative strategies
DEX strategies
etc.
Agent service continously analyzes trending strategies
Agents come to a decision about most profitable course of action
Agents move funds into most profitable strategies
Change strategy
Continue current strategy
AUTONOLAS

Fig. 11: Model of a yield service.

This type of yield service is novel because:

- It does not require that treasury managers constantly optimize between different opportunities, as is the case with existing on-chain yield aggregators.
- In fact, these off-chain yield services could consume on-chain yield aggregators and make their capital even more efficient. The single transaction deployment of capital is much more efficient than current treasury management techniques.
- Yield accrues to the treasury without needing in-house strategists, repetitive votes, or on-chain engineering work to deploy funds.
- Funds can be paid out autonomously for ongoing operational expenses, and new votes can take place to augment or add new modules to the treasury, or simply to remove the funds entirely.
- Security is enforced on-chain for this kind of yield service. A lightweight protocol sits between the service's multisig and the third-party DeFi protocols it is interacting with. As off-chain strategies are piped through this on-chain protocol, actions are sense-checked, allowing strict policies to be enforced regarding where funds can be deployed.

Yield services such as this can expand to cover a multitude of different DeFi opportunities, across protocols and chains. Unlike other yield aggregators, the addition of new strategies is mediated by Autonolas' permissionless and token-governed protocol. We expect this to lead to a proliferation of high-quality components which can be seamlessly – and potentially autonomously – incorporated into competing yield services. Note that these yield services are complementary to, rather than competitive with, on-chain yield aggregators. On-chain yield aggregators with significant brands are well-positioned to deploy and bring this technology to market, and to profit off expanding the strategy component universe.

Trading Strategy Execution

"One point of note that our research has uncovered for current DAOs is the ongoing maintenance of a diverse set of treasury assets."

Achieving yield is only one specific aim of DAO treasury managers. Treasurers are likely to find the protocol's collection of quality, transparent components highly desirable for executing a range of decentralized finance strategies autonomously. Complex hedging, lending, and options strategies, to name a few, are all feasible.

One point of note that our research has uncovered for current DAOs is the ongoing maintenance of a diverse set of treasury assets. When DAOs start up, they often have large quantities of their own tokens, but small quantities of other assets. This is a vulnerability – if there's a significant and extended market drawdown, they can be left without capital to continue developing their project.

To buffer against this, the DAO normally purchases a basket of alternative assets which in aggregate can better weather such extended downturns. An example basket might include 40% DAO's own token, 20% a stablecoin (e.g. USDC or UST), 20% L1 asset (e.g. ETH, JUNO, or SOL), and 20% Bitcoin (or its wrapped versions).

Once the DAO has come to a consensus on the desired portfolio like the one above, what steps do they need to take to build and maintain it? Often these include the following activities, which can be much more efficiently and powerfully done by an autonomous service:

- Asset acquisition
- Portfolio rebalancing

Example: Asset Acquisition

Currently, purchasing this desired set of diverse assets involves analyzing a myriad of trading venues for liquidity to ensure minimal slippage. This is more complicated for large treasuries, but aggregators improve the issue somewhat. They are still fragmented, and rarely aggregate enough liquidity for a single large purchase, which is why the current state of treasury tooling calls for improved aggregators.

Thus far, DAOs have found suboptimal solutions to this problem. Some DAOs deputize subDAOs to act as trading desks, or to vet external teams of traders – both approaches are highly trustful and/or fragile. Some DAOs will simply make large one-off purchases via custom smart contracts, suffering painful slippage. Our intuition suggests that others still simply avoid rebalancing and diversification in general, simply because they do not have an ideal solution.

By contrast, an agent service is well-suited to assist with this initial acquisition:

- DAOs deposit their funds into an on-chain proxy, set their desired portfolio strategies transparently in code (e.g. what size purchases to make, at which venue, at what time, and over what period),
- the agent service then ascertains and executes the optimal path autonomously and transparently.

This initial acquisition process often depends on significant computation that can be handled suitably off-chain. It also relies on a set of components that can interact with trading venues, whose creation is facilitated well by Autonolas' on-chain protocol incentives. Ultimately this means DAOs can move between large financial positions more quickly, with lower overall transaction cost and a superior risk profile.

Example: Portfolio Rebalancing

Once these portfolios are in place, DAOs will also want to maintain their positions. Taking the example target portfolio from above, if the DAO's tokens increase significantly in value (not uncommon for crypto projects), some of those tokens will need to be sold. In order to maintain the desired asset mix, the DAO would need to sell its tokens in exchange for stables, the L1 asset, and Bitcoin.

Autonolas provides an excellent generalized decentralized trade execution environment for maintaining a desired portfolio with significantly less complexity and cost. While the tracking of proportions is not excessively complex, and could be done on-chain, optimizing for gas and slippage requires significant processing with alternative solutions in order to find the best execution across multiple chains. It is also worth noting that through the composability of Autonolas services, treasurers can easily achieve their adjacent goals. For example, treasurers may want to autonomously pay out contributors from their diversified portfolio, which can be set up ahead of time and executed without human input. Similarly, the various assets could be autonomously deployed using a yield service as described above, autonomously and frugally achieving a portfolio that is simultaneously optimally balanced *and* financially productive.

ii. Protocol Parameter Configuration

"Autonolas autonomous services can be used to reliably offload repetitive business operations to reduce costs, increase efficiency, and gain a competitive advantage."

The DAO that manages the parameters of its on-chain protocol has emerged as one of the main types of DAOs. Most protocols have configurable parameters such as the following:

- **Asset Whitelisting**: which assets are whitelisted for use
- **Liquidity Mining Rewards**: how many of the DAO's tokens are paid out to liquidity providers as rewards
- **Dynamic AMM Fees**: whether the protocol charges a fee, and how much it should charge

In almost all cases today, these parameters are adjusted by a whole-DAO vote or by the votes of a subDAO. As with yield, running day-to-day business operations like this introduces massive inefficiency, as well as trust and centralization vectors.

Autonolas autonomous services can be used to reliably offload repetitive business operations to reduce costs, increase efficiency, and gain a competitive advantage. Such a service can be built from scratch, composed of existing components, or simply consumed from an existing provider. The service sources data, evaluates it against a set of DAO-defined criteria, makes a decision about whether to adjust a parameter, and takes action. This action could be to directly update a parameter without any human input, or more commonly to pass a parameter change that can be overturned by vote/dev council during a multi-day timelock.

Below, we briefly examine an Autonolas solution to the first example described above: asset whitelisting.

Example: Asset Whitelisting

Many DeFi protocols have a manual process for proposals, undertaking diligence, and deciding which assets can be used on their platforms. For example, DAOs managing lending protocols like Aave need to decide which assets are safe to use as collateral, and which can be borrowed. Another example: NFT derivative projects like Mimicry need to decide which NFT collections are safe and suitable for trading via their protocol. Whitelists are also particularly relevant for protocols that maintain dynamic indexes of assets. Virtually all categories of DeFi protocol have some form of listing process which decides the class of assets that can be used, according to some set of risk and demand criteria.

It is rare that this is executed with any level of automation, and never fully. Although fulfilling this operation manually can give rise to a high level of security, doing so is uncompetitive compared to the autonomous alternative. Manual execution of this kind of operation results in the following:

- Reduced discovery of desirable assets – it is cost-prohibitive for humans to stay on top of the evolving landscape of assets
- Higher exposure to undesirable assets – when listed assets no longer meet the DAO's criteria, detection and removal will be slower
- Higher cost in maintaining asset whitelists – asset whitelists are often maintained by human participants who have expensive HR and management requirements
- Non-computational interface for machine participants – machine participants (e.g. bots) cannot propose themselves to manual processes, fundamentally reducing their efficacy
- Increased voter apathy – constant repetitive votes reduce voter willingness to participate in asset whitelisting and governance at large
- Difficulty in extending and refining selection criteria – maintaining and extending the set of criteria that governance uses in the whitelisting process is difficult to expand with a manual process
- Difficulty in expanding product offerings horizontally – as protocol DAOs offer more products, each requires parameter adjustment, which becomes unmanageable at best and a risk vector at worst

To sum up, we expect protocol DAOs to be interested in autonomously managing certain parameter configurations because this increases the scalability and robustness of their operations, while simultaneously reducing human frustration, labor cost and fallibility.

b. Protocol Infrastructure

Above, we have elucidated that there are clear use cases for Autonolas agent services in refining the operations of DAOs. We see another major market opportunity for protocols to improve their applications and become "full-stack autonomous".

Many current protocols have fragile off-chain components. These components are built, mostly for lack of superior tooling, to a standard far below on-chain applications. Similar to DAOs, protocols depend on infrastructure delivered by one of the following:

- A set of human participants
- Centralized architecture – e.g. a bot on AWS

- A collection of single-purpose decentralized services
- A custom set of community-run bots

The first two options have obvious downsides in terms of centralization, trustfulness, and management costs. The third option – where protocols rely on services provided by dedicated data sourcing, bridging, indexing, and keeping (often third-party) solutions – is good for specific use cases, but breaks down for more complex, higher-order applications. Finally, custom bots can give protocol developers a large amount of flexibility. However, they require a lot of coordination, can be expensive in terms of token incentives, and provide little guarantee that the service will be reliably executed.

The Autonolas open-source service development framework and on-chain protocol together provide builders with an option that combines out-sourcable, decentralized operation with a composable development model.

Below, we will elucidate some of the use cases for autonomous services in protocol infrastructure, particularly examples of applications and primitives.

i. Applications

We have encountered many applications of Autonolas with existing protocols. Below we dive into two:

- Infrastructure for Cross-chain Yield Aggregation
- Smart Contracts and Configuration-as-a-Service

Example: Infrastructure for Cross-chain Yield Aggregation

Our research suggests that an increasing number of protocols recognize the weakness of the current UX of yield aggregation for individual users. Even within a top yield aggregation app, there is a large number of vaults users need to select and optimize. This creates a heavy ongoing cognitive load for the average DeFi user, reduces their overall return on capital, and increases their risk profile.

We see an opportunity for the emergence of a new class of yield aggregator, which we call the *meta yield aggregator*. Meta yield aggregators – as with the DAO yield service described in the previous section – pull decision-making logic off-chain via an autonomous service. They are thus able to create an extraordinarily simple experience for the end user – depositing assets once to see them get split, propagated, and deployed across chains and protocols according to optimal gas and movement between positions.

These meta yield aggregators will tap into the same reservoir of DeFi components as the DAO yield service, e.g. one component which optimizes earning on Aave, another that optimizes between yield opportunities on an on-chain aggregator (like Solana's

Tulip), and so on.As they do so, and as other types of applications begin to depend on the same set of components, we expect them to radically increase in quantity and depth. This will lead to a shrinking of the flywheel which attracts users through superior yields, providing capital to drive expansion, incorporating more DeFi protocols, and building out better-yielding strategies.

Example: Smart Contracts and Configuration-as-a-Service

An emerging trend with crypto projects is to offer a hybrid service where they deploy a standard set of smart contracts on behalf of another project, and then manage the parameters of those contracts over time.

There are currently teams who operate this sort of service to help other projects manage their liquidity, or to quickly set up and run lending protocols without requiring in-house expertise. We expect this to be an attractive business model.

However, there are technical limitations. As these teams bring on more partners, the complexity of their operations also expands. They need to stay on top of a different set of inputs for each project and cross-check them continuously against their evaluation criteria. This increases their operational costs and likely risks parameter mismanagement at some point, which could lead to a loss of funds and reputational damage.

We expect that such teams will find success in deploying Autonolas autonomous services. These services can watch the same inputs the teams are watching manually, but also cross-check them against criteria and, if desired, can be set up to immediately write parameter changes to the chain in a robust and transparent manner. A secure service written in this way has many advantages. They can be much more responsive to the dynamic environment and can be re-used and updated easily – all with a lower cost and higher reliability than the more manual current processes.

Example: Operating Agent Services for Third Parties

The lifecycle of an agent service that has been described so far relies on the existence of an agent-service owner that is responsible for attracting reliable operators to run the service. Naturally, this approach does not suit every possible entity that wants to run an agent service. It is non-trivial to manage a distributed application or to find the operators to run your service, so you may want to have an agent service of your design running without accepting all the responsibilities of an agent-service owner.

A possible solution to this problem is a *mega agent-service* whose goal is to operate full services or more basic FSM apps on behalf of others. This would allow service owners who do not want to be involved in, for example, choosing and managing agent operators, to code an agent service of their choosing, and the mega agent-service would be in charge of running this service on behalf of their owner (namely, the entity that submitted the code).

ii. Primitives

The Autonolas stack can also be used to build key technological primitives as a substrate for higher-order applications, for example these three supporting higher-order applications outlined above:

- Oracles
- Keepers
- Bridges

Importantly, as these three examples begin to show, since autonomous services built on Autonolas are composable and generalizable by design, more complex and richer infrastructure primitives can be imagined and built than ever before.

Oracles

Blockchains are not able to interact with data and systems outside their native environment. We refer to data and software not on a blockchain as *off-chain*. Most (though not all) on-chain software is so-called “smart contracts”, and relies on off-chain data and services. So-called “oracles” are used to bridge real-world off-chain data with blockchains so that it can be consumed by smart contracts as a source of truth. This enables actions such as clearing buy or sell orders that depend on an asset’s price, or triggering crop insurance payouts depending on weather data.

However, current oracles bringing off-chain data on-chain present several problems :

- *Centralization:* Data providers are usually centralized, which clashes with the decentralization ethos of crypto, is likely less robust due to single points of failure
- *Bias:* Data can be biased. For instance, in the case of an oracle that fetches the price of a given asset, a given exchange could (intentionally or not) offer price data from a particular market that is not representative of the whole market

Decentralized oracle networks help address the issues of centralization and bias. These networks consist of a collection of parties called "nodes" that need to satisfy the following requirements:

- Oracle nodes must extract independent and verified data, so that it is possible to compute various statistics of the different collected values
- Oracle nodes must collectively agree on the observations ultimately used for computing the aggregate value
- Oracle nodes must signal on-chain that they commit to the agreed computed value

In particular, decentralized oracle nodes must be able to communicate and achieve some form of consensus in a distributed and adversarial environment.

Current oracle solutions have important drawbacks. They provide decentralization on-chain (by relying on smart contracts, and therefore on the inherent consensus mechanism of on-chain computation). Carrying out consensus on-chain carries with it a high price of computation and storage, plus the unnecessary costs associated with recording every detail of its workings on-chain (rather than just necessary information).

This limits the number of possible aggregation types you can reasonably carry out. Other providers resort to bespoke solutions that do all of their working out off-chain, yet these are often opaque and centralized, meaning there’s no way of knowing what has been done to the data. To bring costs down, these providers trade decentralization for accuracy and/or security.

Autonolas Oracles

Autonolas oracles are uniquely robust and decentralized due to their multi-agent architecture with off-chain consensus. They are modular, customizable and can be created using the Autonolas oracles toolkit.

Autonolas oracles consist of different and independent agents used to obtain, validate, and aggregate data. Even if some agents go offline, are faulty or malicious, or some provide bad data – the system remains operational due to its fault-tolerant design where agents use a consensus mechanism[28] to agree on their actions. As such, Autonolas oracles can integrate off-chain data to one main chain and stick to blockchain’s promise of decentralization while keeping costs down – they take the fault-tolerance guarantees and consensus mechanisms that power on-chain activity and replicate them off-chain. Since the only items recorded on-chain are the necessary information observed by the oracles and the validation of its correctness, the costs of transactions are reduced without harming robustness and trust-minimization. Moreover, even complex aggregation can be performed off-chain.

This multi-agent architecture also enables agent mutual-verification and blacklisting techniques, as agents are not limited in the logic they can implement. For example, agents could check and detect whether any of their pairs is providing invalid data and stop using them for the final aggregated price. Finally, the agents can implement arbitrarily complex aggregation functions, including machine-learning-based approaches.

“Autonolas oracles can be built in with bespoke features depending on the data requirements (e.g. privacy) of any given application.”

[28] See section on Autonolas technical architecture for more information on consensus gadgets.

Furthermore, because of their modularity, Autonolas' oracles are also highly extensible and can be deployed for a wide range of applications and activities. Plus, unlike some of the more established oracle solutions, Autonolas oracles can be built with bespoke features depending on the data requirements (e.g. privacy) of any given application.

Keepers

Smart contracts enable the secure creation and execution of tasks. The on-chain code that defines a smart contract is triggered when a transaction is executed and some predetermined conditions are met. As such it has a reactive (rather than proactive) nature. Moreover, smart contracts cannot access off-chain data – they rely on oracles to do this. External triggers to execute smart contract code are a hindrance to automation.

Keepers make inroads towards enhancing the power of smart contracts by automating parts of them and allowing an automatic dialogue with the outside world. The term *keeper* refers to an entity that executes a task on behalf of a third party. This entity can be a person (node operator) or a bot.

A task executed by a keeper can be as simple as creating a triggering transaction for the smart contract to execute the transfer of funds when its timer is at a specific time. However, keeper tasks may require a complex off-chain logic, e.g. calling an oracle service to frequently update a given data feed on an L1 blockchain. This off-chain logic is prone to all the off-chain logic issues highlighted in the previous section on oracles.

Autonolas Keepers

As with Autonolas oracles, Autonolas keepers take the fault-tolerance that powers on-chain activity and replicate it off-chain with means to achieve internal consensus. In this vein, Autonolas keepers are naturally aligned with Web3's promise of decentralization and trust-minimization, and can be used to automate more critical tasks. For instance, they could be used to securely control funds and contribute to transaction inputs. We believe that Autonolas keepers offer two *primary benefits*:

- Execution of tasks with fault-tolerance (in contrast to tasks performed by a single keeper bot/human)
- Expansion of the possible use cases (e.g. keepers can be tasked with providing off-chain inputs in the transactions)

Furthermore, as with all services that can be built within Autonolas, Autonolas keepers enjoy a modularized and easily integratable architecture. For instance, Autonolas keepers can leverage complex off-chain logic by calling Autonolas oracles in a decentralized and trust-minimized way, keeping costs down.

As with oracles, Autonolas keepers are fault-tolerant and remain operational even when encountering errors in some agents. In Autonolas keepers, the operators providing the service collaborate rather than compete, so that, depending on levels of redundancy, an Autonolas keeper will have less downtime than a keeper run by a single operator.

As with oracles, in Autonolas keepers the agent instances run a consensus protocol (i.e. a State-Minimized Consensus Gadget) to agree on their actions. In contrast to other existing solutions, this makes Autonolas keepers reliable enough to manage funds. In addition, as with any services created with the Autonolas toolkit, keepers are not limited in the logic they can implement.

Given their modularity and settlement-layer agnosticity, Autonolas keepers can use Autonolas oracles to allow smart contracts to run continuously, to trigger them (by calling complex off-chain logic), and to autonomously fetch data that isn't on the same blockchain.

Bridges

At an abstract level, a bridge can be defined as a system that transfers information between a source chain and a destination chain, where "information" could refer to assets, proofs, or state, to name a few. Most bridge designs [21] have several functions:

- **Monitoring:** There is usually an actor, e.g. an oracle or relayer, that monitors state on the source chain
- **Relaying:** After an actor picks up an event, it needs to transmit information from the source chain to the destination chain
- **Signing:** Actors need to cryptographically sign the information sent to the destination chain

In most of those designs, at least one of the functions is performed off-chain by a single party. This poses the same issues for decentralization and transparency as discussed for oracles and keepers. In addition, relying on a single party to be available at all times in order to perform a bridging functionality is not ideal. For instance, if the relaying function is unavailable, the assets will propagate through the system at a significantly slower pace, even as they incur no additional risk.

Allowing for a network of agents to provide any off-chain bridge functionality currently performed by a single party creates a more robust bridge design with increased availability. This can be achieved naturally using the Autonolas tech stack.

c. Future Use Cases

In the DAO Operations and Protocol Infrastructure sections above, we have focused on the areas in which we expect the stack to gain utility in its early stages. As the ecosystem matures, however, we foresee the emergence of extremely broad-reaching applications. Below, we dive into a few categories of these applications.

i. Full Stack Autonomous Organizations

One of the primary design goals of DAOs is to autonomously coordinate groups of otherwise unaligned entities. Thus far, this has been done by building incentives directly into blockchain protocols to reward or punish entities who interact on their behalf. The most famous example of this is Bitcoin which, broadly speaking, promotes entities that add computing power and security to the network. This model has proliferated through smart contract chains to the application layer – liquidity mining can also be thought of as an evolution of the autonomous coordination concept.

Over the years, DAO coordination mechanisms have dwindled in their capacity to enable fine-grained autonomous direction. This can be seen in the widespread view that DAOs are disorganized and ineffective (as eloquently illustrated in [25]). Tools like SourceCred, as well as emerging projects focused on credentialing, like RabbitHole and Galaxy Project, have begun to push things forward. However, we assert that the primary reason why richer coordination protocols have not emerged is because a decentralized off-chain substrate for building these tools is missing.

To take a rudimentary example, let us consider a protocol that is trying to incentivize their community to promote what they are doing on Twitter. They want to pay out a proportion of their weekly token emissions to people who promote a particular campaign. In order to do this, they need to aggregate the total number of shares using the campaign hashtag, keep a record of every participant's efficacy, and, at the end of the week, submit action to the chain to allocate rewards accordingly.

In isolation, this example seems trivial. But if we extend its depth and breadth, it quickly becomes obvious that this is a powerful way to introduce the benefits of autonomy missing from DAOs. For instance, we could analyze the efficacy of external entities in pushing participants through an entire funnel, across multiple platforms – microblogging, websites, videos, etc. When configured correctly, this begins to resemble a full marketing function, optimizing an entire funnel in a decentralized manner.

Thus far, we have only discussed the narrow marketing function, but it is also worth considering how this thinking could extend to other business functions. A prominent example is the organization's internal operations, whether for code contribution or for community moderation via Discord-based signaling games and rule enforcement. Organizations can combine this with internal processes like financial accounting, which maintain exact proportions of funds in accordance with projected runways, and which fulfill payroll autonomously.

Strategy and orientation are critical operations to all organizations. Orientation involves evaluating the effectiveness of business units and campaigns in making progress towards organizational objectives. We expect that, before long, applications will be able to cross-check the output of coordination components to hone and push towards organizational objectives with total autonomy. Note that organizations that are managed entirely autonomously will remain in the realm of science fiction for a long time. But it is reasonable to expect that certain functions will gradually become fully autonomous within a reasonable amount of time.

With these internal functional and strategic operations in place, we begin to glimpse a full-stack autonomous organization – that is an organization that has begun to eliminate the need for human management.

Further to the examples above, it is possible to imagine full services which are produced, sold, and consumed autonomously, DAO to DAO. We have talked in previous sections about the production of services, such as the examples of liquidity management and outsourced implementation of lending protocols. It is also interesting to consider the possibility that DAOs could detect an emerging internal need, perform a search of DAOs solving that need, and then negotiate, establishing discounts and service-level agreements, and finally consume the external service – all entirely autonomously. As DAO to DAO interaction becomes more autonomous, we begin to witness the emergence of an entirely new class of machine-only economy, which has the potential to radically expand the productive capacity of society.

It is worth reiterating that the properties of the Autonolas stack and protocol are what make the above examples possible. Without the composable software stack, it would be prohibitively slow to build out every component of a new autonomous service operation. The same composability makes it possible for DAOs to interact and enable a machine-machine economy. Without robustness, decentralization, and transparency, teams and users would not entrust their critical operations to external services. All these features work together to enable this new building block for a future generation of increasingly-autonomous organizations.

ii. The Everything API

As the set of components and services in the Autonolas ecosystem expands, the reach of supported functionality and data could make a versatile interface with extensive Web3 coverage, fuelled by the application flywheel and in-built incentives.

To paint a picture of how this could begin to look, we can focus on DeFi. As the meta yield aggregators (described in a prior section) take shape, one side effect is a rich set of robust DeFi components. Although initially built for the yield use case, the components are open for consumption via the on-chain protocol. For example, if a meta yield aggregator builds the underlying architecture for pulling funds in and out of lending protocols, or for trading on a derivatives exchange, these components are instantly available for other developers to use. At a tipping point where the component set is rich enough, application developers are enabled to enter the market and build entire trading platforms upon it, all via a single and relatively simple integration into an Autonolas service.

As this pattern of a single use case generating a tipping point of early components replicates itself in other verticals such as NFTs and gaming, an increasingly rich and wide-reaching base of functionality and data is established. All of that functionality and data can in principle be accessed through a very small set of Autonolas service endpoints. This creates a compelling experience and Schelling point for developers who are looking to build powerful higher-order applications with little interest in building or maintaining the requisite middleware.

5. Governance

The Autonolas DAO is made up of holders of veOLAS. In this section, we introduce the key aspects of governance of the Autonolas protocol, including the governance process, key features of Autonolas' governance architecture, and the importance of the community's role in governance.

a. Role of Governance

The Autonolas governance system is designed to assume various control points to steer the Autonolas protocol. Examples of governance control points include the following:

- Whitelist of LP tokens permitted for bonding
- Change of the parameters of the control mechanism that incentivizes bonding
- Weight developers and treasury's share of donations
- Change the parameters for donation payouts and top-ups
- Manage donors blacklist
- Deployment of Protocol-owned Liquidity to yield-bearing investment opportunities
- Allocate grants from Autonolas treasury
- Pause and unpause code registries
- Fully specify and manage Autonolas PoSe
- Tune sensible governance parameters (e.g. minimal voting right necessary to make a governance proposal, quorum fraction to accept a proposal)
- Fully upgrade tokenomics, governance, and registries contracts

Autonolas tokenomics and its analysis are a work in progress and are subject to change. The Autonolas DAO can fine-tune aspects of the protocol over time. One example, including enabling Level 3 of the tokenomics, e.g. enriching DCMs with ICMs.

Autonolas will keep the core governance coordination mechanisms, in particular OLAS and veOLAS, anchored on a single chain. However, due to Autonolas' cross-chain focus, some minimal governance modules will need to be deployed on other chains.

Governance also limits protocol forking risk by incentivizing a growing community and ecosystem to rally around the Autonolas protocol rather than creating forks of it.

Forking risks can be reduced by having high community buy-in, easy entry into the community, and continued innovation.

To avoid facilitating donations from illicit or nefarious activities and to prevent any distribution of incentives for illicit or nefarious activities by the protocol, governance may blacklist donors.

b. The Governance Process

Building on the experience of existing decentralized protocols, such as Compound [22], we envision three distinct components for governance:

- The veOLAS virtualized claim on OLAS
- A governance module
- Timelock

Together, these components allow the community to propose, vote on, and implement changes. Proposals can notably modify system parameters, support new technological directions, and add entirely new functionality to the protocol.

veOLAS holders have non-delegable and non-transferable voting rights. Any address that holds a certain *governance-threshold* number of veOLAS can create a governance proposal for voters to vote upon.

When a governance proposal is created, it enters a *review period*, after which voting weights are recorded and voting begins. Voting lasts for a certain number of *election duration* days. If a majority of at least the approval *threshold* of votes is cast for the proposal, it is queued in the Timelock, and can be implemented several days later, called the *preparation period*. In other words, Timelock adds a delay for governance decisions to be executed and the governance workflow requires a queue step before execution.

Exceptionally, some changes to the Autonolas Protocol can be executed by a *community-owned multisig wallet*, bypassing the governance process. This allows a set of trusted actors to overrule governance in certain aspects without governance discussion, e.g. a security exploit needing to be patched.

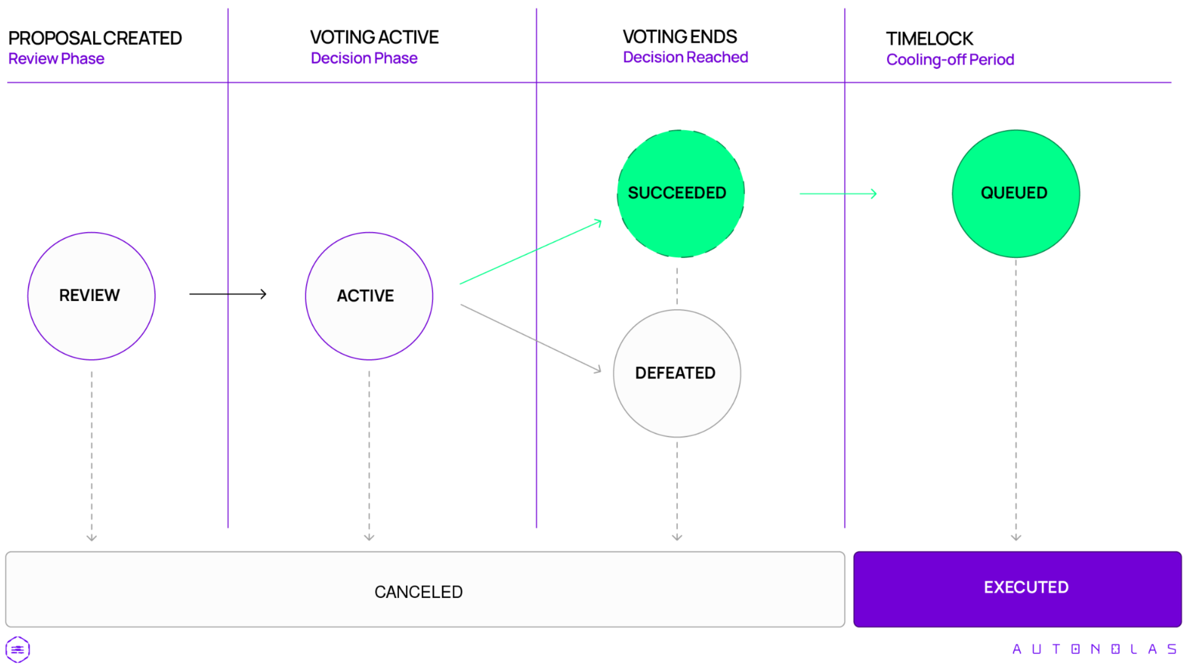

Fig. 12: The Autonolas voting process.

Optionally, the on-chain proposal stage can be preceded by an off-chain signaling vote, for example on platforms like Snapshot. Off-chain signaling can be a good tool to gauge the interest of community participants who have a low willingness to pay for on-chain voting, and who might otherwise be unable to express their preferences.

c. Governance Architecture

i. Governance Module

Inspired by Compound governance architecture [22,23] and based on OpenZeppelin modular system of governance contracts architecture [24], the following set of smart contracts is part of the Autonolas Protocol governance module:

veOLAS contract

This contract is inspired by the Voting Escrow contract from Curve [26]. Any party that wants to participate in a vote in governance needs to hold the virtualized token veOLAS. To obtain veOLAS it is required to lock some amount of OLAS. The veOLAS tokens received by locking OLAS are assigned a certain voting weight in the Autonolas DAO. The voting weight is both time (length of the locking period) and amount (quantity of tokens locked) weighted.

GovernorOLAS contract

This contract is used to manage the DAO voting functionality and it is derived from the OpenZeppelin modular system of governance contracts [24]. Any address that meets the proposal threshold may propose a governance action. Users may vote for, against, or abstain. Any vote in favor of a given proposal will count towards the quorum, the percentage of the total veOLAS supply allowing the proposal to pass. If the quorum is reached, the proposal will be queued in the timelock before being executed.

Timelock

This contract is used in conjunction with the GovernorOLAS contract. Its purpose is to implement a delay in the execution of an already approved governance action. It is derived from the OpenZeppelin TimelockController contract [24].

We highlight some aspects of this architecture below:

Non-Upgradeable

The GovernorOLAS contract implements the core mechanics of the governance module. It is a non-upgradeable contract. If it ever needs upgrading, to adapt to future community needs, or to fix bugs, the governor can be replaced via a governance vote (using the governor to be replaced).

Administrative powers

Secure access controls play a significant role in the security of governance modules, which often have sensitive parameters. The contract allows an administrator account to change several sensitive parameters: the voting period, the proposal threshold, and the voting delay.

⋆ **Note**: The administrator account is the Timelock contract, so changes to governance itself are time-delayed.

ii. Other contracts related to Autonolas governance module

OLAS contract

This is an ERC-20 token contract with Solmate [27] optimization and some long-term inflation constraints to give further guidance to governance processes.

Community multisig

Some changes to the Autonolas Protocol will not go through governance. Instead, they will be executed by a multisig wallet made up of a set of trusted actors. It is expected from the multisig trustees that they limit this power to a set of specific, sensitive changes in the protocol, which need to be applied without being able to rely on the formal governance process. This allows bypassing governance in certain aspects without discussion, e.g. a security exploit that needs patching.

The community multisig has the necessary administrative roles to initiate, execute, and cancel proposals. This multisig is authorized by the Timelock contract and is subject to the community-defined minimum time delay. After the time delay has passed, the multisig proposal can be executed.

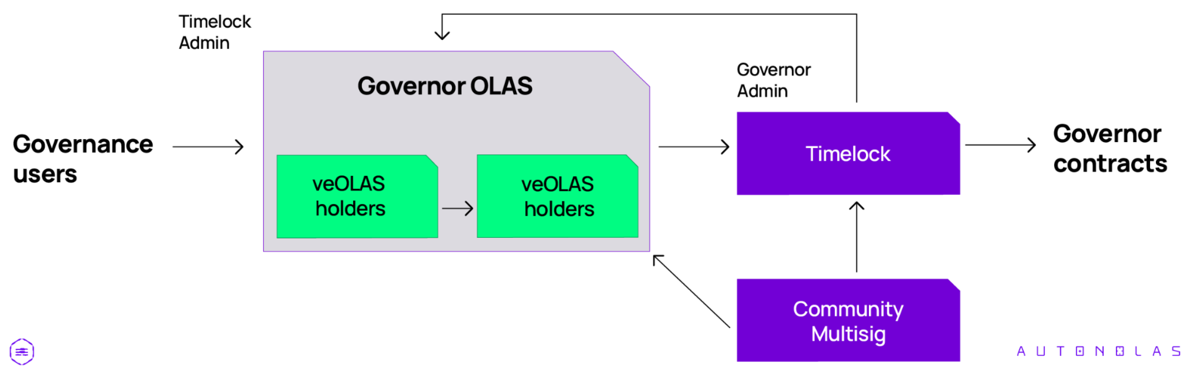

Fig. 13: The Autonolas governance architecture.

d. Community Building

Over the long term, and as the community grows, the protocol will transition to being managed in an increasingly decentralized manner. It is therefore important that Autonolas cultivate early on a group of people who are passionate, loyal, and who can understand the system as well as govern it.

Additionally, the growth and success of Autonolas will largely depend on attracting and maintaining strong developer talent. As highlighted earlier in this document, the technical capabilities of Autonolas and its economic incentives are key components of its success. However, facilitating good relations between developers and generally cultivating a constructive social environment are equally as critical.

Developer talent is only one pillar, however; a diversity of talent beyond the purely technical will additionally bolster Autonolas' long-term competitive advantage. Beyond developers, Autonolas seeks to attract and nurture creatives, researchers, marketers, analysts, and more.

Over and above established practices of community-building in crypto, Autonolas will need to experiment with novel approaches. One example is Alter Orbis, which is an open world whose lore is designed around the network's main narratives. Each narrative was crafted to educate and convince potential community members of Autonolas mission.

With the lore provided as initial building blocks, from its inception Alter Orbis is already open to any community member to extend – an open sandbox that is engaging for creators, and that helps spread the underlying message and priorities of the network.

Beyond Alter Orbis, Autonolas will constantly evaluate emerging technologies and community-building best practices in order to maximize community strength and engagement.

References

[1] *Exploring the concept of "autonomy" in technology-enabled future:*

a digest of thinking and works. Kelsie Nabben.
https://kelsienabben.substack.com/p/exploring-the-concept-of-autonomy

[2] *DAOs, DACs, DAs and More: An Incomplete Terminology Guide* Vitalik Buterin, 6 May 2014
https://blog.ethereum.org/2014/05/06/daos-dacs-das-and-more-an-incomplete-terminology-guide/

[3] *Technological Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages*. Carlota Perez. Edward Elgar Pub (2003)

[4] *The Curse of Bigness: Antitrust in the New Gilded Age*. Tim Wu. Columbia Global Reports (2018)

[5] *Placeholder Thesis Summary.* Placeholder VC.
https://ipfs.io/ipfs/QmZL4eT1gxnE168Pmw3KyejW6fUfMNzMgeKMgcWJUfYGRj/Placeholder%20Thesis%20Summary.pdf

[6] *Open-aea framework*. https://valory-xyz.github.io/open-aea/

[7] *SoK: Consensus in the Age of Blockchains*. Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn and George Danezis. 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019. https://arxiv.org/abs/1711.03936

[8] *What is Tendermint?* Tendermint Inc.
https://docs.tendermint.com/master/introduction/what-is-tendermint.html

[9] *Scalable and Probabilistic Leaderless BFT Consensus through Metastability.* Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, Emin Gün Sirer, 24 Aug 2020, arXiv:1906.08936v2

[10] *Solana: A new architecture for a high performance blockchain*. Anatoly Yakovenko.
https://solana.com/solana-whitepaper.pdf

[11] *Lachesis: Scalable Asynchronous BFT on DAG Streams.* Quan Nguyen, Andre Cronje, Michael Kong, Egor Lysenko, Alex Guzev, 5 August 2021.
https://arxiv.org/pdf/2108.01900.pdf

[12] *Towards Trustworthy DeFi Oracles: Past, Present and Future.* Y. ZHAO, X. KANG, T. LI, C.-K. CHU, H. WANG. https://arxiv.org/abs/2201.02358

[13] Introducing the Cross-Chain Interoperability Protocol (CCIP) Chainlink. 5 August 2021.

[14] *4 eras of blockchain computing: degrees of composability.* Jesse Walden. 12 December 2018.
https://jessewalden.com/4-eras-of-blockchain-computing-degrees-of-composability/

[15] A Primer on Bonding. Olympus DAO. 20 February 2021.
https://olympusdao.medium.com/a-primer-on-oly-bonds-9763f125c124

[16] *"Due to a major AWS outage…".* @dYdX.
https://twitter.com/dYdX/status/1468293558360805381

[17] *Organizations.* DeepDAO. https://deepdao.io/organizations

[18] *Gnosis Safe*. @tschubotz. https://dune.xyz/tschubotz/gnosis-safe_2

[19] *How to use Aragon Agent*. Aragon.
https://hack.aragon.org/docs/guides-use-agent

[20] *Lido and Unslashed create a new standard: insurance as an integrated solution.* Marouane Hajji. 23 February 2021.
https://medium.com/unslashed/lido-and-unslashed-create-a-new-standard-insurance-as-an-integrated-solution-e9b3406f3136

[21] *Blockchain Bridges: Building Networks of Cryptonetworks*. Dmitriy Berenzon. Medium, 8 September 2021.
https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8

[22] *Governance.* Compound Labs Inc. https://compound.finance/docs/governance

[23] *Smart Contract Security Guidelines #4: Strategies for Safer Governance systems.* OpenZeppelin.
https://blog.openzeppelin.com/smart-contract-security-guidelines-4-strategies-for-safer-governance-systems/

[24] *OpenZeppelin Governance docs*
https://docs.openzeppelin.com/contracts/4.x/governance

[25] *"DAOs".* @ASvanevik.
https://twitter.com/ASvanevik/status/1505521318648721411?s=20&t=449ACaTsP0smDk0FdOoLgQ

[26] *Voting Escrow contract. Curve Finance.*
https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/VotingEscrow.vy

[27] *ERC-20 token contract. Solmate.*
https://github.com/transmissions11/solmate/blob/main/src/tokens/ERC20.sol

Glossary

AEA – *Autonomous Economic Agent*

CID – *Content Identifiers*

DAO – *Decentralized Autonomous Organization*

DCM - *Direct Contribution Measure*

DEX – *Decentralized Exchange*

DLT – *Distributed Ledger Technologies*

ICM – *Indirect Contribution Measures*

**OLAS – ***Autonomolas Native Token*

subDAO – *subset/subgroup of a DAO*

UCF – *Useful Code Factor*

USF – *Useful Service Factor*

veOLAS – *Non-tradeable and non-delegable virtualized claim of locked OLAS*

Valory is the VC-backed team of engineers, researchers and commercial thinkers that has created the Autonolas stack

---

Official website

autonolas.network

Twitter

@autonolas

Discord

discord.com/invite/z2PT65jKqQ

APPENDIX

a. The Concept of Global Time

Timeouts are checked against the "global clock" obtained from the timestamps on the blocks provided by the consensus engine. For example, consider a block b_1 with timestamp t_1, and block b_2 with timestamp $t_2 = t_1 = 10s$. Assume that after block b_1 is processed by the ABCI Application, we are in the first round, and that this first round can trigger a timeout event with timeout T=5s. At the time that block b_2 gets confirmed, the timeout event was already triggered (since $t_1 + T < t_1 + t_2$), and the associated transition is already taken in the FSM, regardless of the content of block b_2.

b. Composition of Finite-State Machines

To enable the composition of services from constituent and reusable parts the chaining of ABCI applications is supported. The concept of Chain of ABCI applications or concatenated FSMs can informally be described as follows. Let ABCIApp1 and ABCIApp2 be two ABCI applications. Let ABCIApp3 be obtained by combining ABCIApp1 and ABCIApp2. Then the states and the transitions among states of the FSM defined by ABCIApp3 are described as follows:

1. The initial state of ABCIApp3 is the initial state of ABCIApp1.
2. The non-initial and non-final states of ABCIApp3 are the disjoint union of the non-initial and non-final states of ABCIApp1 and non-final states of ABCIApp2.
3. The final state of ABCIApp3 is the final state of ABCIApp2.
4. Any transition from a non-final state in ABCIApp1 to a non-final state in ABCIApp1 is not modified.
5. Any transition from non-final states in ABCIApp1 going to a FINAL STATE in ABCIApp1 goes to the initial state of ABCIApp2.
6. The transitions between internal states of ABCIApp2 are not modified.

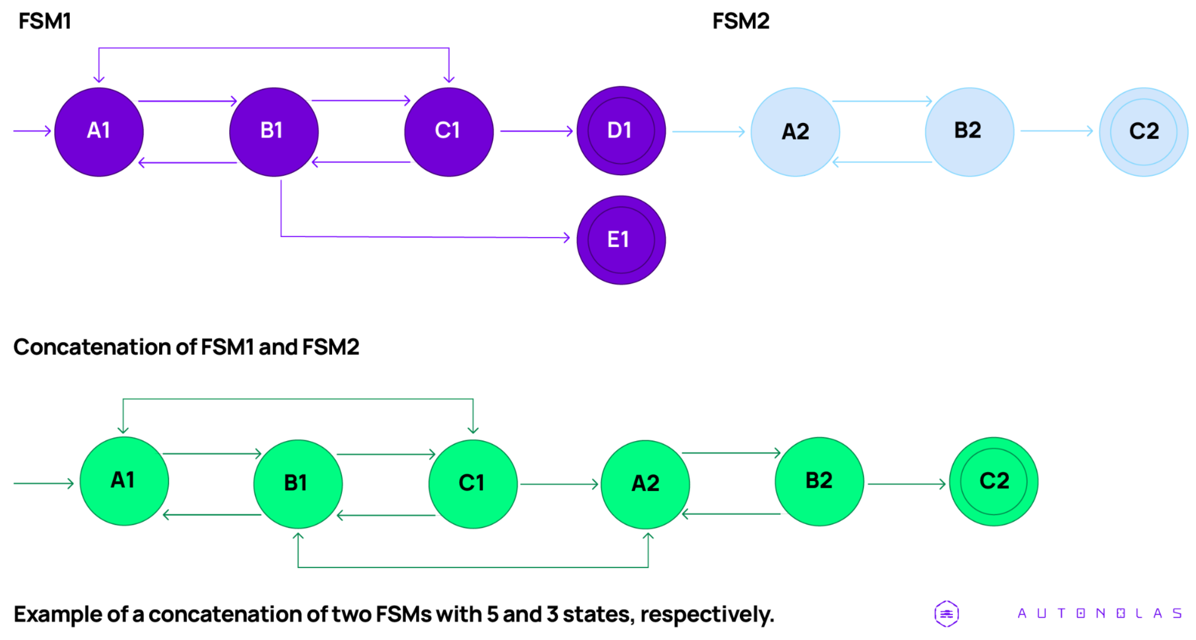
FSM1
FSM2
A1
B1
C1
D1
E1
A2
B2
C2
Concatenation of FSM1 and FSM2
A1
B1
C1
A2
B2
C2
Example of a concatenation of two FSMs with 5 and 3 states, respectively.
AUTONOLAS

Fig. 14: Example of a concatenation of two FSMs with 5 and 3 states, respectively.

c. On-chain Referencing of Off-chain Code

The on-chain protocol has no means of validating anything off-chain. Hence, the protocol optimistically assumes that any code referenced in it is correct. This assumption is reinforced through the tokenomics: components, agents, or service configurations that are incorrectly referenced are unusable and therefore not showing up in the reward mechanism. Furthermore, if a service owner misspecified the mapping on the contract and config level then this should be obvious to operators in the service who can then choose to flag it or ignore it (if benign)

When a component or agent is minted the developer is required to provide a hash. This hash refers to the metadata as per the ERC-721 metadata standard. It has the following form:

```
{
  "title": "Asset Metadata",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Identifier of the component/agent"
```

```
        },
        "description": {
            "type": "string",
            "description": "Describes the component/agent"
        },
        “image”: {
            "type": "string",
            "description": "A URI pointing to a resource with mime type image/*
representing the asset to which this NFT represents. Consider making any images at a
width between 320 and 1080 pixels and aspect ratio between 1.91:1 and 4:5 inclusive."
        },
        "code_uri": {
            "type": "string",
            "description": "An IPFS URI pointing to the component/agent code.."
        },
       "attributes": {
            "type": "object",
            "description": "Defines a set of attributes which classifies and defines the NFT.
            Allows for applications to build filters and attribute search functionalities for a
            set of NFTs.",
        },
    }
}
```

The metadata file itself then points to the underlying code via the “code_uri” field. The `attributes` object can be used to express optional attributes like versions.

Similarly, the service has a config hash. This also points to a metadata field as above:

```
{
    "title": "Asset Metadata",
    "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Identifier of the service"
    },
    "description": {
      "type": "string",
      "description": "Describes the service"
    },
    “code_uri”: {
      "type": "string",
      "description": "An IPFS URI pointing to the service code."
    },
   "attributes": {
      "type": "object",
      "description": "Defines a set of attributes which classifies and defines the NFT.
      Allows for applications to build filters and attribute search functionalities for a
      set of NFTs.",
    },
  }
}
```

This config file, referenced indirectly on the service, can then point to the exact versions of the NFTs used (i.e. primary or secondary hashes).

d. Code Versioning and Updating

Code has bugs and occasionally needs to be updated with new features. The protocol allows developers to update their component and agent NFTs. The ERC-721 standard does not permit updating of the primary hash. Therefore, Autonolas extends the ComponentRegistry and AgentRegistry to allow developers to append new hashes on-chain. An “update” method on the ERC-721 accepts a new metadata hash, which is then stored on the contract without replacing the previous hash. This can act as a signaling device for the developer that new code is available for the component. The protocol does not enforce how versioning is implemented. We expect the ecosystem to naturally converge on a number of conventions to enable the interoperability of components.

The dependency structure of a component, once enshrined on-chain, cannot be updated. When the dependency structure changes the developer can create a new NFT on-chain representing the changed component.

e. Tokenomics – Potential Future Features

This whitepaper introduces version 1 of the Autonolas protocol. After launch, governance will likely continue innovating on the protocol. To this end, the Autonolas Improvement Proposal process (AIP) will be created. This will be modeled after the Ethereum Improvement Proposal (EIP) process.

f. Protocol-owned Components

The protocol can purchase components or agents from developers via governance. Protocol-owned components can act as a mechanism for developers to take money off the table if they need to. Since protocol-owned components are approved by governance, the risk of low-value code draining OLAS is mitigated. Autonolas stakeholders can value components on Level 3.