NEBA Token Architecture

Contents

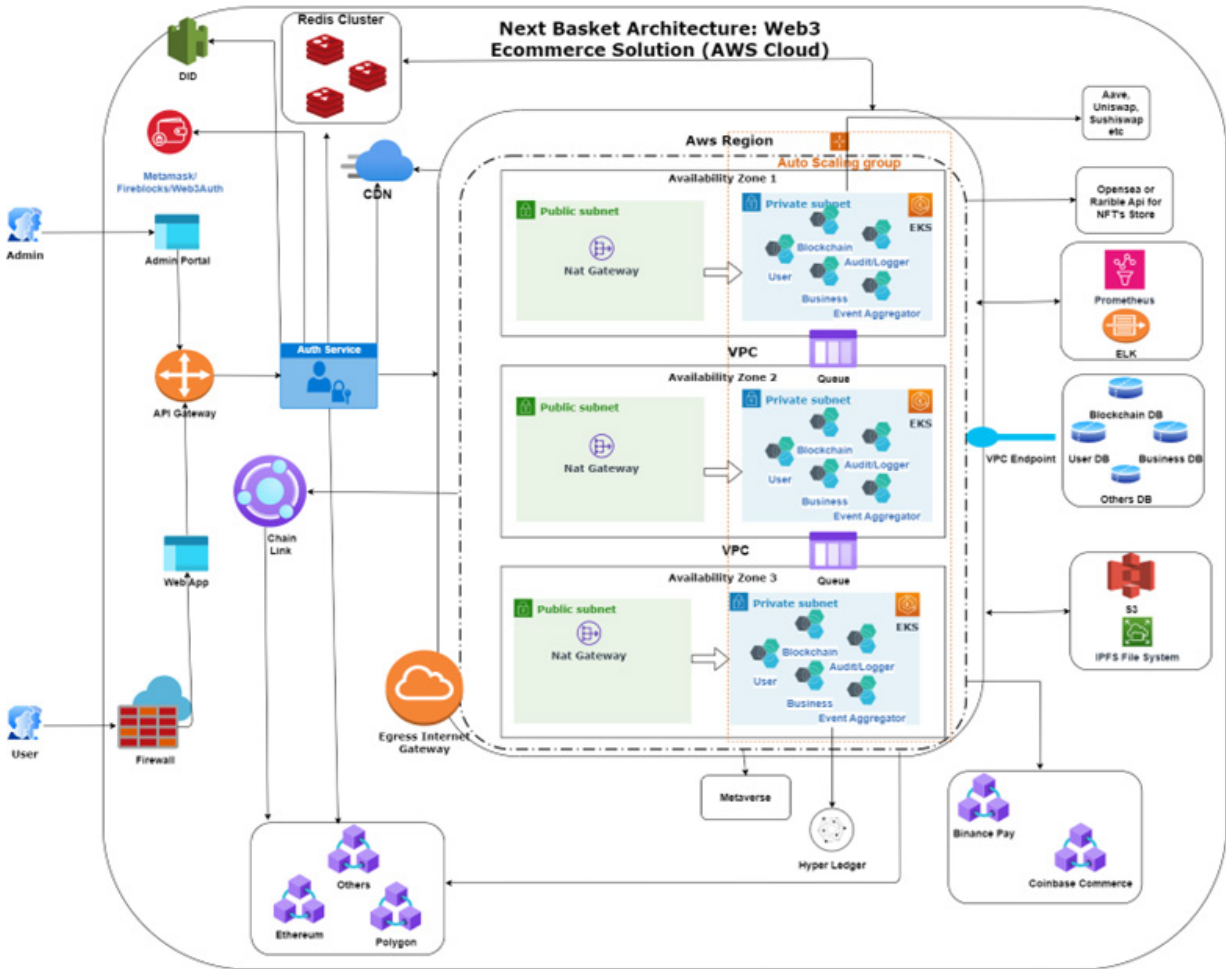
1. Preface

NEXT BASKET is an innovative platform designed to revolutionize e-commerce by blending Web3 functionalities with traditional online store management. The platform aims to give clients the ability to create and manage decentralized online stores, integrating cutting-edge blockchain technologies (e.g., Ethereum, Polygon, and Arbitrum) with a scalable, microservices-based infrastructure. Through its native utility token, NEXT BASKET ensures a seamless e-commerce experience by incorporating decentralized identity management, cryptocurrency payments, and carefully structured tokenomics.

2. System Architecture Overview

The overall architecture of the product is structured to support a hybrid of traditional e-commerce functionalities and Web3 solutions. It includes:

- **Microservices** for essential business logic, broken down for scalability and resilience.
- **Blockchain Integration** for identity, payments, and decentralized data management.
- **Front-End** interfaces (e.g., React) for user interactions.
- **Decentralized Storage** (e.g., IPFS, Filecoin) for permanent or high-availability content.
- **Messaging and Oracles** for off-chain data feeds and decentralized notifications.



3. Infrastructure on AWS

- **Compute:**
  - **Amazon EC2** for microservices deployment. Amazon EC2 for microservices deployment.
- **Containerization:**
  - **Docker** and **Kubernetes** for deploying and managing microservices.
- **Database:**
  - **Amazon RDS** for structured data,
  - **Amazon DynamoDB** for key-value operations,
  - **Decentralized databases** like OrbitDB for certain Web3 data needs.
- **Storage:**
  - **Amazon S3** for general storage, integrated with IPFS or Arweave for critical decentralized data.
- **CI/CD:**
  - **AWS CodePipeline** for automated deployments, supporting blue-green releases.
- **Monitoring:**
  - **AWS CloudWatch** plus the **ELK Stack** for real-time logging and analytics.
- **Networking:**
  - **AWS API Gateway** for secure API management.
  - **CloudFront CDN** for caching and content delivery.
- **Caching:**
  - **Redis** for session storage and data caching.
- **Security and Segregatio**n:
  - **VPC & NAT Gateway** for isolating services within AWS.
  - **AWS WAF** or **Cloudflare** for firewall and traffic filtering.
- **Event-Driven Com**munication:
  - **SQS/RabbitMQ/Kafka** for asynchronous communication among services.

4. Tools and Technologies

- **Frontend:**
  - **React** with **Redux** or other state management.
- **Backend:**
  - **Node.js** with Express or **NestJS.**
- **Block**chain Integration:
  - Ethereum mainnet, Polygon, or Arbitrum for Layer 2 scaling.
- **Security:**
  - **AWS Secrets Manager,** IAM policies, and smart contracts for identity and payment validation.
  - Data Storage:
  - S3, RDS, IPFS, and additional blockchain-based storage as needed.
- **Wallet Integrations:**
  - **MetaMask, Fireblocks, Web3Auth, etc.**

5. Microservices Architecture

Technologies: Docker, Kubernetes, Istio (Service Mesh)

- Blue-Green Deployments
  - **Overview:** Maintain two environments—"Blue" (current production) and "Green" (new version)—to ensure zero-downtime upgrades.
  - **Usage in NEXT BASKET:**
    - Critical for deploying new features in payment services, identity management,
      or smart contract integration without interrupting active sessions.
    - Quickly revert to the old environment if issues arise.
  - **Integration:**
    - Orchestrated via Kubernetes, which manages traffic switching.
    - AWS CodePipeline/CodeDeploy orchestrates the deployment process.
- Service Mesh with Istio
  - **Overview:** Istio provides a layer for managing service-to-service traffic, enhancing observability and security.
  - **Usage in NEXT BASKET:**
    - Fine-grained control over communications between services (e.g., load balancing, traffic policies).
    - Simplified debugging with tracing, logging, and metrics.

- **Integration:**
  - Sidecars (Envoy) run next to each microservice, controlling traffic.
  - Administrators can configure traffic routing, fault injection, and rate-limiting centrally.

6. Logging and Monitoring

Technologies: ELK Stack, Prometheus, Blockchain Analytics

- **Centralized Logging with ELK Stack**
  - **Overview:** ELK (Elasticsearch, Logstash, Kibana) centralizes log collection, storage, and visualization.
  - Usage in NEXT BASKET:
    - Aggregates logs from microservices, APIs, and Kubernetes, providing a unified view.
    - Facilitates root-cause analysis through dashboards and searches.
  - **Integration:**
    - Logstash collects logs from different sources (including blockchain interactions).
    - Elasticsearch indexes them for quick searches, while Kibana is used for data visualization.

- **Blockchain Analytics for Smart Contract Transactions**
  - **Overview:** Tracks on-chain actions like token transfers, contract executions, and transaction failures.
  - **Usage in NEXT BASKET:**
    - Monitors token distribution, staking, and suspicious activities.
  - **Integration:**
    - Data from Etherscan or similar APIs is pulled into the ELK stack or custom dashboards.
    - Provides real-time insights into the blockchain aspects of the platform.

7. Admin Panel with Role-Based User Access (RBUA)

The admin panel is pivotal for configuring and supervising the NEXT BASKET platform. It offers granular role-based controls, letting administrators handle platform operations efficiently and securely.

7.1. Features of the Admin Panel

- **Role-Based User Access (RBUA)**
  - **User Roles:**
    - **Super Admin:** Full access, including system settings and tokenomics.
    - **Admin:** Access to core business areas (orders, support) with limited configuration rights.
    - **Merchant Admin:** Restricted to their own store data and product management.
    - **Support Staff:** Handles user issues, ticketing, and limited actions.
    - **Auditor:** Read-only access for compliance and audits.
  - **Permissions:**
    - Each role has customizable CRUD rights on users, tokens, orders, products, etc.
    - Dynamic updates—no code changes needed to alter permissions.
  - **Backend Enforcement:**
    - Middleware checks roles before granting access to sensitive endpoints.

- **System Configuration**
  - **Blockchain Settings**: Set up Layer 1 and Layer 2 networks (addresses, gas fees).
  - **Tokenomics Management**: Adjust token issuance, staking, distribution, and monitor supply metrics.
  - **Storage Management:** Configure IPFS, Filecoin, etc., and track usage.
  - **Messaging and Notifications**: Configure Waku or XMTP and notification templates.

- **User and Merchant Management**
  - **User Management:** Create, edit, or deactivate user accounts; view activity logs.
  - **Merchant Management:** Approve merchant onboarding; configure store-level settings.

- **Monitoring and Analytics**
  - **Real-Time Dashboards:** Transaction throughput, token operations, response times.
  - **Transaction Logs:** Detailed logs of on-chain and off-chain activities.
  - **Reports:** Customizable reports for usage, performance, and financial metrics.
- **Security and Compliance**
  - **Access Logs:** Tracks admin activity, configuration changes, login attempts.
  - **Audit Tools:** Ensures GDPR or AML compliance; encryption for sensitive operations.
  - **Secure Access:** Two-factor authentication for admin accounts; end-to-end encryption on critical data.
- **Customization and Localization**
  - **Interface Customization:** Skins, layouts, role-based content.
  - **Multi-Language Support:** Adapts for global user bases.

7.2. Technologies and Architecture

- **Frontend:**
  - React.js for an interactive UI: Redux or Context API for state.
  - Material-UI or Ant Design for polished interfaces.
- **Backend:**
  - Node.js (Express/NestJS) for admin APIs.
  - GraphQL (optional) for flexible queries.
  - Custom RBAC (Role-Based Access Control) middleware.
- **Database:**
  - PostgreSQL/MySQL for relational data.
  - Redis for caching frequently accessed data (roles, permissions).
- **Authentication/Authorization:**
  - OAuth 2.0 or OpenID Connect for identity.
  - JWT tokens for stateless sessions.
- **Logging and Monitoring:**
  - ELK stack for admin-side logs.
  - Prometheus + Grafana for backend metrics.
- **Deployment and Scalability:**
  - Kubernetes handles container orchestration.
  - CI/CD pipelines automate testing and deployment.

7.3. Integration with Platform Components

- **Blockchain:**
  - Admins can modify blockchain settings (network addresses, fees).
  - On-chain analytics for token and smart contract monitoring.
- **Microservices:**
  - REST/GraphQL endpoints to coordinate identity, payments, storage, etc.
  - RBAC ensures secure microservice calls.
- **Messaging:**
  - Admins configure Waku topics and user notification preferences.
- **Logging/Analytics:**
  - Aggregation in dashboards for compliance and real-time oversight.

7.4. Benefits

- **Configurable and Secure:** Fine-grained control paired with robust encryption and auditing.
- **Efficient:** Streamlines operations for token management, user roles, and platform metrics.
- **Scalable:** Modern frameworks ensure it can handle large spikes in traffic.

8. Security

Technologies: AWS IAM, Smart Contract Audits, Encryption

- **IAM Roles and Policies on AWS**
  - **Overview: A**WS IAM restricts resource access based on least-privilege principles.
  - **Usage:** Assign roles to EC2, Lambda, or container tasks, controlling who can access S3, RDS, etc.
  - **Integration**: Secrets stored in AWS Secrets Manager; only authorized roles can retrieve them.
- **Smart Contract Audits**
  - **Overview:** Audits detect vulnerabilities in token, escrow, and identity contracts.
  - **Usage:** Tools like CertiK, OpenZeppelin, or Trail of Bits validate code before deployment.
  - **Integration:** Automated CI/CD pipelines run checks with MythX or Slither for early issue detection.

- **Data Encryption and Security Standards**
  - **Overview:** Encryption protects data at rest and in transit.
  - **Usage:** TLS for all microservice communications; data in RDS or S3 encrypted using AWS KMS.
  - **Integration:** AWS Certificate Manager handles SSL/TLS for external endpoints.
- **Integration with Other Components**
  - **Frontend:** TLS encryption ensures secure interactions.
  - **Blockchain:** On-chain analytics integrated with security monitoring for suspicious activity detection.
  - **Kubernetes:** Service mesh polices secure inter-microservice traffic.
- **Use Cases in NEXT BASKET**
  - **Zero-Downtime Updates**: Blue-green deployments for new token features without affecting live users.
  - **Proactive Issue Resolution:** Centralized logs and metrics help identify and resolve performance bottlenecks.
  - **Granular Access Control:** IAM ensures only authorized identities interact with sensitive systems.
  - **Transparency:** Audited smart contracts build trust with stakeholders.

9. Decentralized Identity Management

9.1. DID-Based Authentication Using Standards Like OAuth

- **What is DID?**
  - Decentralized Identifiers are self-sovereign, independent digital IDs. The DID subject controls it without a central authority.
  - The W3C DID standard ensures consistent structure and verifiability.
- **How OAuth Fits**
  - OAuth is an open standard for authorization, commonly used to grant external apps access to user data without sharing passwords.
  - By integrating DID with OAuth, the platform can validate user identities on-chain while leveraging OAuth tokens for session management.
- **Integration with the Architecture**
  - DID records may be stored on Ethereum or Arbitrum, ensuring immutability.
  - OAuth flows use DID-based credentials, verified via DID-compatible identity providers (e.g., uPort, Verifiable Credentials).

9.2. Integration of Decentralized Key Management Systems (DKMS)

- **What is DKMS?**
  - A framework for users to manage their private keys without relying on centralized entities.
  - Tools like MetaMask or uPort store keys locally, giving users full control.
- **How It Integrates**
  - DKMS handles cryptographic operations for identity verification and transaction signing.
  - For instance, a user's purchase transaction is signed with the private key in their local wallet, and the contract verifies it on-chain.
- **Architecture Integration**
  - MetaMask or other wallets connect to the React frontend.
  - Node.js microservices verify the transaction on Ethereum or similar networks.

9.3. Smart Contracts for Access Control and Authentication

- **Access Control**
  - Role-based access logic can be embedded in contracts.
  - Merchants might have elevated privileges, while customers only hold basic transaction rights.
- **Authentication**
  - Smart contracts validate DID-based credentials.
  - Ensures only users with valid on-chain identities can execute certain platform actions.
- **Integration**
  - Deployed primarily on Ethereum, Polygon, or Arbitrum.
  - The frontend and backend coordinate to facilitate transaction signing and data verification.

9.4. End-to-End Integration of Tools

- **Frontend (React):**
  - Users log in with wallets or identity providers.
  - Actions (e.g., buying an item) prompt signature requests that are broadcast on-chain.
- **Backend (Node.js):**
  - Enforces business logic, bridging the blockchain with internal services.
  - Interacts with smart contracts to confirm user permissions before finalizing actions.

- **Blockchain:**
  - Processes identity checks, token transactions, and updates to user roles.
  - Layer 2 solutions (e.g., Polygon, Arbitrum) reduce costs.

10. Smart Contracts and Tokenization

10.1. Detailed Explanation

Smart contracts and tokenization form the core of NEXT BASKET's decentralized commerce. By using Solidity and standards like ERC-20, ERC-721, and ERC-1155, the platform automates payments, ownership transfers, and asset management in a trustless environment.

10.2. Tools Recommended and Their Role

- **Solidity**
  - Primary language for Ethereum and EVM-compatible chains.
  - Implements logic for escrow, staking, and payments.
- **ERC-20**
  - Fungible tokens used for utility or governance.
  - Supports payments, rewards, and partial ownership.
- **ERC-721 (NFT)**
  - Non-fungible tokens for unique items, e.g., digital certificates, special product badges.
- **ERC-1155**
  - Multi-token standard blending fungible and non-fungible functionalities.
- **DEX APIs**
  - Integrate with Uniswap or 1inch for liquidity and token swaps.

10.3. Workflow in Greater Detail

- **Merchant Payments**
  - Payment contracts handle funds, storing them in escrow until delivery is confirmed.
- **Asset Tokenization**
  - Inventory, loyalty rewards, or certificates minted as NFTs.
  - References stored on IPFS to ensure metadata integrity.

- **Token Economy Management**
  - Utility token powers transactions, staking rewards, and governance.
- **DEX Integration**
  - Allows on-platform token swaps or price discovery.
  - Chainlink oracles can provide price feeds for real-time valuation.

10.4. Integration with Other Components

- **Blockchain Networks:**
  - Ethereum (mainnet) for security, Polygon/Arbitrum for scaling.
- **Backend Services:**
  - Node.js microservices trigger or listen to events (e.g., token transfers).
- **Frontend:**
  - React UI allows users to manage tokens, stake, or mint NFTs.
- **Payment Systems:**
  - Smart contracts tie into crypto gateways for frictionless payments.

11. Cryptocurrency Payments

11.1. Detailed Explanation

NEXT BASKET's crypto payment system lets customers pay in various digital assets, while merchants can seamlessly receive, hold, or convert them. By combining decentralized escrow, profit-sharing contracts, and automated fiat conversion, it ensures a user-friendly experience.

11.2. Tools Recommended and Their Role

- **Coinbase Commerce**
  - Direct gateway to accept major cryptocurrencies (BTC, ETH, stablecoins).
  - Webhooks let the platform know when payments are completed.
- **Binance Pay**
  - Expands the range of accepted cryptocurrencies.
  - Facilitates multi-party split payments (e.g., profit sharing).
- **Layer 2 Solutions (Optimistic Rollups, etc.)**
  - Handle high-volume or microtransactions at reduced costs.
- **Smart Contracts (Escrow, Payout Distribution)**
  - Securely lock funds until conditions are met.

- **Crypto-to-Fiat Conversion**
  - Automates conversions, safeguarding merchants from volatility.

11.3. Workflow in Greater Detail

- **Initialization**
  - Customer selects “pay with crypto.”
  - The system generates a payment address or QR code via Coinbase/Binance Pay.
- **Transaction Processing**
  - Once funds arrive, a webhook notifies the backend.
  - Escrow contract holds the payment until the platform confirms product delivery.
- **Escrow and Distribution**
  - On confirmation, the contract disburses funds to merchants or splits among partners.
- **Layer 2 Scaling**
  - For small or frequent transactions, the system uses Optimistic Rollups to reduce gas.
  - Regular batch settlements on Ethereum for final security.
- **Conversion and Compliance**
  - If merchants want fiat, an automatic process converts crypto into USD/EUR.
  - AML/KYC procedures are enforced by the platform’s compliance layer.

11.4. Integration with Other Components

- **Blockchain:**
  - Final settlement on Ethereum or a chosen Layer 2.
- **Microservices:**
  - Payment microservice orchestrates gateways, escrow, and conversion.
- **Frontend:**
  - Users see real-time payment confirmations and status updates.
- **Logging:**
  - Payment events recorded in the ELK stack and chain analytics.

12. Decentralized Storage

12.1. Tools Recommended and Their Role

- **IPFS**
  - Content-addressed storage for product images, documents, or metadata requiring immutability.
- **Arweave**
  - Permanent data storage; suitable for regulatory archives.
- **Filecoin**
  - Incentivized storage for large datasets or redundancy.
- **Amazon S3**
  - Centralized object storage for temporary or non-critical data.
- **Relational/NoSQL Databases**
  - For structured data (orders, user profiles) and references to decentralized files.

12.2. Workflow in Greater Detail

- **Uploading and Indexing**
  - Files are uploaded via the frontend, hashed, and pinned on IPFS or stored in Filecoin.
  - Database references the hash or storage ID for quick retrieval.
- **Permanent Archiving (Arweave)**
  - Important data is encrypted and stored on Arweave for immutable, permanent access.
- **Hybrid S3 + Decentralized Storage**
  - S3 for quick-access caching, IPFS/Filecoin for redundancy and trustlessness.

12.3. Integration with Other Components

- **Blockchain:**
  - Smart contracts reference IPFS/Arweave hashes.
- **Frontend:**
  - Retrieves file data from IPFS nodes or fallback (S3).
- **Microservices:**
  - Orchestrate user uploads, retrieval, and hashing.
- **Logging:**
  - Monitoring usage, ensuring content is pinned and accessible.

13. Messaging Network

13.1. Detailed Explanation

A decentralized messaging network underpins communication between services and users. By using protocols like XMTP, Waku, or LibP2P, NEXT BASKET gains secure, private, and fault-tolerant channels.

13.2. Tools Recommended and Their Role

- **XMTP**
  - Peer-to-peer, end-to-end encrypted messages, often using wallet-based identities.
- **Waku (Whisper v2)**
  - Pub/Sub for decentralized broadcasts and light node support.
- **LibP2P**
  - Underlying framework for secure peer discovery and communication.

13.3. Workflow in Greater Detail

- **Message Initialization**
  - A service or user sends an encrypted message to a topic or a specific recipient.
- **Publishing and Subscribing**
  - Nodes subscribe to relevant topics, receiving only matching messages.
- **Routing**
  - LibP2P handles finding peers, ensuring stable connections.
- **Optional Persistence**
  - XMTP can store/replay messages if the receiver is offline.

13.4. Integration with Other Components

- **Blockchain**
  - Signed data can be transmitted for contract interactions (e.g., dispute resolution).
- **Frontend**
  - Wallet-based messaging for user support or notifications.
- **Microservices**
  - Communicate over decentralized channels for internal coordination.
- **Logging/Analytics**
  - Non-sensitive metadata can be collected centrally.

13.5. Use Cases in NEXT BASKET

- **Order Updates:** Real-time notifications on shipment or payment status.
- **User Support:** Secure user-service chat.
- **Token Economy Alerts:** Broadcast changes in staking rewards or governance proposals.

14. Integration of Oracles

14.1. Detailed Explanation

Oracles like Chainlink bring off-chain data on-chain, enabling automated triggers for shipping events, dynamic pricing, or external market feeds.

14.2. Tools Recommended and Their Role

- **Chainlink**
  - Fetches data from external APIs (e.g., crypto prices, shipment trackers).
  - Aggregation ensures reliability and mitigates single-point failures.
- **Node Operators**
  - Decentralized services that retrieve and verify data.
- **Chainlink VRF**
  - Provides provably fair random numbers for loyalty rewards or prize events.
- **Validation Mechanisms**
  - Aggregation contracts weed out anomalies, ensuring accurate final values.

14.3. Workflow in Greater Detail

- **Data Request**
  - A smart contract calls Chainlink for a price or shipping update.
- **Fetching Data**
  - Multiple node operators gather info from various sources.
- **Aggregation**
  - Chainlink aggregates results, discarding outliers.
- **Delivery to Smart Contracts**
  - Contract receives validated data, triggering next steps (e.g., releasing escrow).

14.4. Integration with Other Components

- **Smart Contracts**
  - Rely on timely, accurate data for condition-based logic.
- **Payment Systems**
  - Use real-time exchange rates to finalize user transactions.
- **Frontend**
  - Displays oracle-fed updates (e.g., location tracking).
- **APIs and Logistics**
  - Connect logistics provider data to Chainlink for on-chain confirmation.

14.5. Use Cases in NEXT BASKET

- **Dynamic Token Pricing:** Real-time price feeds for stable valuations.
- **Delivery Confirmation:** Automated escrow release upon proof-of-delivery.
- **Randomized Rewards:** Chainlink VRF for fair distribution in loyalty programs.

15. Layer 2 for Optimization and Scaling

15.1. Detailed Explanation

Layer 2 solutions handle the limitations of Ethereum's mainnet—namely, high gas and limited throughput—by offloading computations and batching transactions.

15.2. Tools Recommended and Their Role

- **Polygon**
  - High-speed sidechains for microtransactions, NFT minting.
- **Arbitrum**
  - Rollups to reduce fees and increase throughput, suitable for payment settlements.
- **Optimism**
  - Optimistic rollups for subscription or bulk order processing.

15.3. Workflow in Greater Detail

- **Transaction Offloading**
  - Users or merchants interact on Layer 2, significantly lowering costs.
- **Batch Processing**
  - Rollup solutions bundle many transactions, submitting them periodically to Ethereum.

- **Cross-Layer Assets**
  - Utility tokens can be bridged between L1 and L2.
- **Settlement**
  - Finalizes on Ethereum to retain security guarantees.

15.4. Integration with Other Components

- **Blockchain Layer**
  - Smart contracts on both mainnet and L2.
- **Smart Contracts**
  - Staking, rewards, or escrow operate at lower cost on L2.
- **Microservices**
  - Payment services route smaller transactions to L2.
- **Frontend**
  - Guides users to deposit/withdraw tokens across layers.

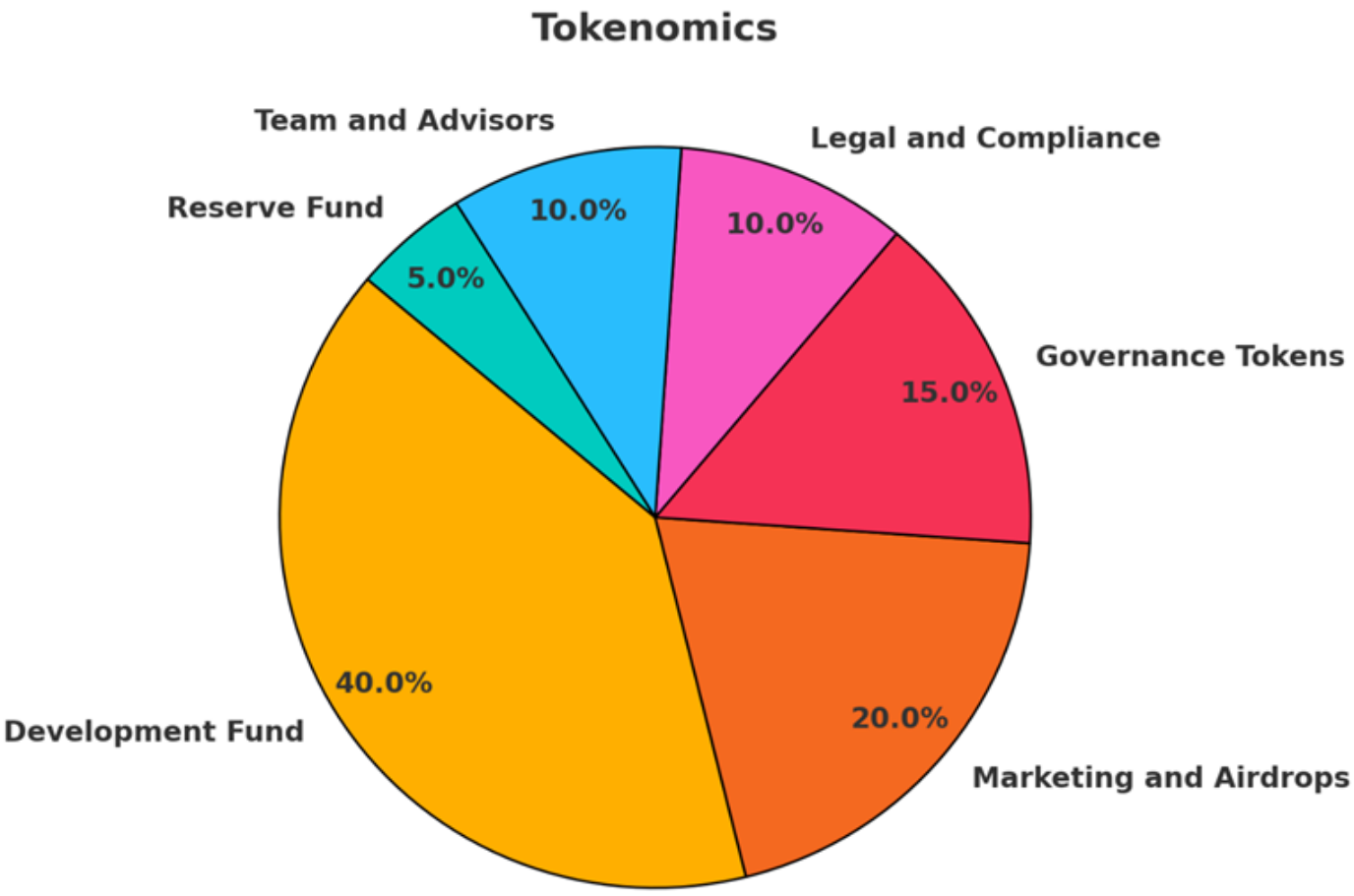
15.5. Use Cases in NEXT BASKET

- **Cost-Effective Payments:** Minimal fees for day-to-day transactions.
- **High-Frequency Rewards:** Handling loyalty points or multi-step staking.
- **NFT Marketplaces:** Bulk minting and trading with reduced gas.
- **Order Settlements:** Process large volumes on L2 for efficiency.

16. Tokenomics Design

(No changes applied; this section remains as originally presented.)

- **Token Allocation**
  - Development Fund: 40%
  - Marketing and Airdrops: 20%
  - Governance Tokens: 15%
  - Legal and Compliance: 10%
  - Team and Advisors: 10%
  - Reserve Fund: 5%
- **Token Utilities**
  - Payments, staking, governance, loyalty points.
  - Governance tokens for community-driven decisions.
  - Reserve tokens for exchange liquidity.



Final Note

This revised document provides improved clarity, consistent terminology, and comprehensive details about the NEXT BASKET Web3 architecture. It does not alter any of the tokenomics or allocation figures—only the text around the architecture and related processes has been edited for correctness and clarity.