Here’s a single copy-paste prompt you can drop into Replit AI to build the complete Axiom plus KeyGrow system on peaq, end-to-end. It will wipe old contracts, create the new suite, wire constructors, add deploy scripts that record addresses into addresses.json, grant roles, compile, and deploy.

# Replit AI prompt starts below

Goal Rebuild the Axiom Protocol smart contract suite for peaq EVM as a clean Hardhat project. Remove any existing contracts and scripts. Create the new contracts listed here, add deployment scripts that mirror a consistent pattern, write deployed addresses into a single addresses.json, and wire initial role grants. Then compile and provide ready-to-run commands. Network is peaq EVM (chainId 3338).

Project setup

1. If there is any existing contracts/, scripts/, or deployments/ content, delete it entirely.
2. Initialize a Hardhat project if missing.
3. Install dependencies: npm i @openzeppelin/contracts @nomicfoundation/hardhat-toolbox dotenv
4. Create a .env template with the following keys: PEAQ_RPC=https://evm.peaq.network
   DEPLOYER_PK=0xYOUR_PRIVATE_KEY
   DISTRIBUTION_VAULT=0xVAULT_ADDRESS_TO_RECEIVE_INITIAL_AXM
   ADMIN=0xYOUR_MULTISIG_OR_ADMIN
   RAF_SINK=0xREAL_ESTATE_FUND_WALLET
   DEPIN_TREASURY_SINK=0xDEPIN_TREASURY_WALLET
   LIQUIDITY_SINK=0xLIQUIDITY_WALLET
   TREASURY_SINK=0xGENERAL_TREASURY_WALLET
   KYC_SCHEMA=0x0000000000000000000000000000000000000000000000000000000000000000
   ACCREDITED_SCHEMA=0x0000000000000000000000000000000000000000000000000000000000000000
   NON_US_SCHEMA=0x0000000000000000000000000000000000000000000000000000000000000000
   LICENSE_SCHEMA=0x0000000000000000000000000000000000000000000000000000000000000000

Hardhat config Create or replace hardhat.config.js with:

```
require("dotenv").config();

require("@nomicfoundation/hardhat-toolbox");



const { PEAQ_RPC, DEPLOYER_PK } = process.env;



module.exports = {

  solidity: {

    version: "0.8.20",

    settings: { optimizer: { enabled: true, runs: 500 } }

  },

  networks: {
```

```
    peaq: {
      url: PEAQ_RPC || "https://evm.peaq.network",
      chainId: 3338,
      accounts: DEPLOYER_PK ? [DEPLOYER_PK] : []
    }
  }
};
```

Contract suite Create these Solidity files under contracts/. All contracts are constructor-wired, non-upgradeable, and use AccessControl where relevant. Comments use slashes only.

1. AXM.sol Fixed supply governance token with Permit and Votes. Mints total supply to DISTRIBUTION_VAULT at deployment.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract AXM is ERC20, ERC20Permit, ERC20Votes {
    uint256 public constant TOTAL_SUPPLY = 15_000_000_000 ether;
    event InitialSupplyMinted(address indexed vault, uint256 amount);

    constructor(address distributionVault)
        ERC20("Axiom", "AXM")
        ERC20Permit("Axiom")
    {
        require(distributionVault != address(0), "vault is zero");
        _mint(distributionVault, TOTAL_SUPPLY);
        emit InitialSupplyMinted(distributionVault, TOTAL_SUPPLY);
    }

    function _afterTokenTransfer(address from, address to, uint256 amount)
```

```
        internal override(ERC20, ERC20Votes)
    { super._afterTokenTransfer(from, to, amount); }

    function _mint(address to, uint256 amount)
        internal override(ERC20, ERC20Votes)
    { super._mint(to, amount); }

    function _burn(address from, uint256 amount)
        internal override(ERC20, ERC20Votes)
    { super._burn(from, amount); }
}
```

2. EmissionsVault.sol Linear vesting of AXM to a beneficiary.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract EmissionsVault is ReentrancyGuard {
    IERC20 public immutable axm;
    address public immutable beneficiary;
    uint64  public immutable start;
    uint64  public immutable duration;
    uint256 public released;

    event Funded(address indexed from, uint256 amount);
    event Released(address indexed to, uint256 amount);

    constructor(address _axm, address _beneficiary, uint64 _start, uint64 _duration) {
        require(_axm != address(0) && _beneficiary != address(0), "zero addr");
        require(_duration > 0, "duration=0");
        axm = IERC20(_axm);
```

```
        beneficiary = _beneficiary;
        start = _start;
        duration = _duration;
    }

    function topUp(uint256 amount) external nonReentrant {
        require(amount > 0, "amount=0");
        require(axm.transferFrom(msg.sender, address(this), amount), "transferFrom fail");
        emit Funded(msg.sender, amount);
    }

    function releasable() public view returns (uint256) {
        return _earned() - released;
    }

    function release() external nonReentrant {
        uint256 amt = releasable();
        require(amt > 0, "nothing");
        released += amt;
        require(axm.transfer(beneficiary, amt), "transfer fail");
        emit Released(beneficiary, amt);
    }

    function _earned() internal view returns (uint256) {
        uint256 bal = axm.balanceOf(address(this));
        uint256 total = bal + released;
        if (block.timestamp <= start) return 0;
        if (block.timestamp >= start + duration) return total;
        return (total * (block.timestamp - start)) / duration;
    }
}
```

3. DIDRegistry.sol Verifiable credential hashes with issuer, expiry, and revoke.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";

contract DIDRegistry is AccessControl {
    bytes32 public constant REGISTRAR_ROLE = keccak256("REGISTRAR_ROLE");
    bytes32 public constant ISSUER_ROLE    = keccak256("ISSUER_ROLE");

    struct Credential {
        bytes32 hash;
        address issuer;
        uint64  issuedAt;
        uint64  expiry;
        bool    revoked;
    }

    mapping(address => mapping(bytes32 => Credential)) public creds;

    event Issued(address indexed subject, bytes32 indexed schema, bytes32 hash, address
issuer, uint64 expiry);
    event Revoked(address indexed subject, bytes32 indexed schema);
    event Extended(address indexed subject, bytes32 indexed schema, uint64 newExpiry);

    constructor(address admin) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(REGISTRAR_ROLE, admin);
    }

    function issue(address subject, bytes32 schema, bytes32 vcHash, uint64 expiry) external
onlyRole(ISSUER_ROLE) {
        require(subject != address(0) && vcHash != bytes32(0), "bad args");
        creds[subject][schema] = Credential({
```

```
            hash: vcHash,
            issuer: msg.sender,
            issuedAt: uint64(block.timestamp),
            expiry: expiry,
            revoked: false
        });
        emit Issued(subject, schema, vcHash, msg.sender, expiry);
    }

    function revoke(address subject, bytes32 schema) external {
        Credential storage c = creds[subject][schema];
        require(c.issuer == msg.sender || hasRole(REGISTRAR_ROLE, msg.sender), "not
issuer/registrar");
        require(!c.revoked, "revoked");
        c.revoked = true;
        emit Revoked(subject, schema);
    }

    function extend(address subject, bytes32 schema, uint64 newExpiry) external {
        Credential storage c = creds[subject][schema];
        require(c.issuer == msg.sender || hasRole(REGISTRAR_ROLE, msg.sender), "not
issuer/registrar");
        require(!c.revoked, "revoked");
        require(newExpiry >= c.expiry, "shorter");
        c.expiry = newExpiry;
        emit Extended(subject, schema, newExpiry);
    }

    function isValid(address subject, bytes32 schema) external view returns (bool) {
        Credential memory c = creds[subject][schema];
        return c.hash != bytes32(0) && !c.revoked && (c.expiry == 0 || block.timestamp <=
c.expiry);
    }
}
```

4. ComplianceAdapter.sol Stateless checks for KYC, accredited, non-US.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;


import "@openzeppelin/contracts/access/AccessControl.sol";

import "./DIDRegistry.sol";


contract ComplianceAdapter is AccessControl {

    DIDRegistry public immutable did;

    bytes32 public immutable KYC_SCHEMA;

    bytes32 public immutable ACCREDITED_SCHEMA;

    bytes32 public immutable NON_US_SCHEMA;


    constructor(address admin, address didRegistry, bytes32 kyc, bytes32 accredited, bytes32
nonUs) {

        _grantRole(DEFAULT_ADMIN_ROLE, admin);

        did = DIDRegistry(didRegistry);

        KYC_SCHEMA = kyc;

        ACCREDITED_SCHEMA = accredited;

        NON_US_SCHEMA = nonUs;

    }


    function checkBasicKYC(address user) public view returns (bool) {

        return KYC_SCHEMA == bytes32(0) ? true : did.isValid(user, KYC_SCHEMA);

    }


    function checkAccredited(address user) public view returns (bool) {

        return ACCREDITED_SCHEMA == bytes32(0) ? true : did.isValid(user,
ACCREDITED_SCHEMA);

    }


    function checkNonUS(address user) public view returns (bool) {
```

```
        return NON_US_SCHEMA == bytes32(0) ? true : did.isValid(user, NON_US_SCHEMA);
    }

    function allowPublicSale(address user) external view returns (bool) {
        return checkBasicKYC(user);
    }

    function allowRegD(address user) external view returns (bool) {
        return checkBasicKYC(user) && checkAccredited(user);
    }

    function allowRegS(address user) external view returns (bool) {
        return checkBasicKYC(user) && checkNonUS(user);
    }
}
```

5. NodeLicenseNFT.sol Soulbound license for node operators.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

contract NodeLicenseNFT is ERC721, AccessControl {
    bytes32 public constant ISSUER_ROLE = keccak256("ISSUER_ROLE");
    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
    uint256 public nextId = 1;
    mapping(uint256 => bytes32) public metadataCid;
    bool public soulbound = true;

    event Minted(uint256 indexed tokenId, address indexed to, bytes32 cid);
    event Burned(uint256 indexed tokenId);
    event SoulboundSet(bool enabled);
```

```
    event MetadataSet(uint256 indexed tokenId, bytes32 cid);

    constructor(address admin) ERC721("Axiom Node License", "AXN") {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(ISSUER_ROLE, admin);
        _grantRole(MANAGER_ROLE, admin);
    }

    function setSoulbound(bool enabled) external onlyRole(MANAGER_ROLE) {
        soulbound = enabled;
        emit SoulboundSet(enabled);
    }

    function mint(address to, bytes32 cid) external onlyRole(ISSUER_ROLE) returns (uint256 id)
{
        require(to != address(0), "zero addr");
        id = nextId++;
        _safeMint(to, id);
        if (cid != bytes32(0)) {
            metadataCid[id] = cid;
            emit MetadataSet(id, cid);
        }
        emit Minted(id, to, cid);
    }

    function burn(uint256 tokenId) external onlyRole(MANAGER_ROLE) {
        _burn(tokenId);
        emit Burned(tokenId);
    }

    function setMetadata(uint256 tokenId, bytes32 cid) external onlyRole(MANAGER_ROLE) {
        require(_exists(tokenId), "no token");
        metadataCid[tokenId] = cid;
```

```
        emit MetadataSet(tokenId, cid);
    }

    function _beforeTokenTransfer(address from, address to, uint256 tokenId, uint256 batchSize)
internal override {
        super._beforeTokenTransfer(from, to, tokenId, batchSize);
        if (soulbound && from != address(0) && to != address(0)) revert("soulbound");
    }
}
```

6. DePINNodeRegistry.sol Registers nodes, checks DID license schema.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "./DIDRegistry.sol";

contract DePINNodeRegistry is AccessControl {
    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
    DIDRegistry public immutable did;
    bytes32 public immutable LICENSE_SCHEMA;

    enum Status { Inactive, Active }

    struct Node {
        address operator;
        bytes32 deviceDid;
        bytes32 licenseHash;
        bytes32 metadataCid;
        uint96  weight;
        Status  status;
    }
```

```
    uint256 public nextId = 1;
    mapping(uint256 => Node) public nodes;
    mapping(address => uint256[]) public byOperator;

    event Registered(uint256 indexed id, address indexed operator);
    event StatusSet(uint256 indexed id, Status status);
    event WeightSet(uint256 indexed id, uint96 weight);
    event MetadataUpdated(uint256 indexed id, bytes32 metadataCid);

    constructor(address admin, address _did, bytes32 licenseSchema) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(MANAGER_ROLE, admin);
        did = DIDRegistry(_did);
        LICENSE_SCHEMA = licenseSchema;
    }

    function register(address operator, bytes32 deviceDid, bytes32 licenseHash, bytes32
metadataCid) external returns (uint256 id) {
        require(operator != address(0) && deviceDid != bytes32(0), "bad args");
        require(did.isValid(operator, LICENSE_SCHEMA), "license invalid");
        id = nextId++;
        nodes[id] = Node({ operator: operator, deviceDid: deviceDid, licenseHash: licenseHash,
metadataCid: metadataCid, weight: 1, status: Status.Active });
        byOperator[operator].push(id);
        emit Registered(id, operator);
        emit StatusSet(id, Status.Active);
        emit WeightSet(id, 1);
    }

    function setStatus(uint256 id, Status s) external onlyRole(MANAGER_ROLE) {
        nodes[id].status = s;
        emit StatusSet(id, s);
    }
```

```
    function setWeight(uint256 id, uint96 w) external onlyRole(MANAGER_ROLE) {
        nodes[id].weight = w;
        emit WeightSet(id, w);
    }

    function setMetadata(uint256 id, bytes32 cid) external onlyRole(MANAGER_ROLE) {
        nodes[id].metadataCid = cid;
        emit MetadataUpdated(id, cid);
    }

    function operatorNodes(address op) external view returns (uint256[] memory) {
        return byOperator[op];
    }
}
```

7. DePINRewardRouter.sol Epoch settlement and AXM payouts to node operators.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./DePINNodeRegistry.sol";
import "./DIDRegistry.sol";

contract DePINRewardRouter is AccessControl, ReentrancyGuard {
    bytes32 public constant ORACLE_ROLE = keccak256("ORACLE_ROLE");
    IERC20 public immutable axm;
    DePINNodeRegistry public immutable registry;
    DIDRegistry public immutable did;
    bytes32 public immutable KYC_SCHEMA;
```

```
    mapping(uint256 => bool) public epochSettled;
    mapping(uint256 => uint256) public accrued;

    event Funded(address indexed from, uint256 amount);
    event Settled(uint256 indexed epoch, uint256 nodes, uint256 total);
    event Claimed(uint256 indexed nodeId, address indexed to, uint256 amount);

    constructor(address admin, address _axm, address _registry, address _did, bytes32
_kycSchema) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(ORACLE_ROLE, admin);
        axm = IERC20(_axm);
        registry = DePINNodeRegistry(_registry);
        did = DIDRegistry(_did);
        KYC_SCHEMA = _kycSchema;
    }

    function topUp(uint256 amount) external nonReentrant {
        require(amount > 0, "amount=0");
        require(axm.transferFrom(msg.sender, address(this), amount), "transferFrom fail");
        emit Funded(msg.sender, amount);
    }

    function settleEpoch(uint256 epoch, uint256[] calldata nodeIds, uint256[] calldata scores,
uint256 rewardPool) external onlyRole(ORACLE_ROLE) {
        require(!epochSettled[epoch], "epoch done");
        require(nodeIds.length == scores.length, "length mismatch");
        require(rewardPool > 0, "pool=0");
        require(axm.balanceOf(address(this)) >= rewardPool, "insufficient");
        uint256 totalWeightScore = 0;
        for (uint256 i = 0; i < nodeIds.length; i++) {
            (, , , , uint96 weight, DePINNodeRegistry.Status s) = registry.nodes(nodeIds[i]);
            if (s == DePINNodeRegistry.Status.Active && scores[i] > 0 && weight > 0) {
```

```
                totalWeightScore += scores[i] * uint256(weight);
            }
        }
        require(totalWeightScore > 0, "no eligible");
        uint256 totalAccrued = 0;
        for (uint256 i = 0; i < nodeIds.length; i++) {
            (, , , , uint96 weight, DePINNodeRegistry.Status s) = registry.nodes(nodeIds[i]);
            if (s != DePINNodeRegistry.Status.Active || scores[i] == 0 || weight == 0) continue;
            uint256 share = (rewardPool * (scores[i] * uint256(weight))) / totalWeightScore;
            accrued[nodeIds[i]] += share;
            totalAccrued += share;
        }
        epochSettled[epoch] = true;
        emit Settled(epoch, nodeIds.length, totalAccrued);
    }

    function claim(uint256 nodeId, address to) external nonReentrant {
        DePINNodeRegistry.Node memory n = registry.nodes(nodeId);
        require(n.operator == msg.sender, "not operator");
        if (KYC_SCHEMA != bytes32(0)) {
            require(did.isValid(n.operator, KYC_SCHEMA), "kyc invalid");
        }
        uint256 amt = accrued[nodeId];
        require(amt > 0, "nothing");
        accrued[nodeId] = 0;
        require(axm.transfer(to, amt), "transfer fail");
        emit Claimed(nodeId, to, amt);
    }
}
```

8. AXMStakingPool.sol Time-based AXM rewards for stakers.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
```

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

contract AXMStakingPool is AccessControl, ReentrancyGuard {
    bytes32 public constant FUNDER_ROLE = keccak256("FUNDER_ROLE");
    IERC20 public immutable axm;

    uint256 public rewardRate;
    uint256 public lastUpdate;
    uint256 public rewardPerTokenStored;

    uint256 public totalStaked;
    mapping(address => uint256) public balances;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;

    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardAdded(uint256 amount, uint256 newRate, uint256 duration);
    event RewardClaimed(address indexed user, uint256 amount);

    constructor(address admin, address axmToken) {
        require(admin != address(0) && axmToken != address(0), "zero addr");
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(FUNDER_ROLE, admin);
        axm = IERC20(axmToken);
        lastUpdate = block.timestamp;
    }

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
```

```
        lastUpdate = block.timestamp;
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalStaked == 0) return rewardPerTokenStored;
        return rewardPerTokenStored + ((rewardRate * (block.timestamp - lastUpdate) * 1e18) /
totalStaked);
    }

    function earned(address account) public view returns (uint256) {
        return (balances[account] * (rewardPerToken() - userRewardPerTokenPaid[account]) /
1e18) + rewards[account];
    }

    function stake(uint256 amount) external nonReentrant updateReward(msg.sender) {
        require(amount > 0, "amount=0");
        totalStaked += amount;
        balances[msg.sender] += amount;
        require(axm.transferFrom(msg.sender, address(this), amount), "transferFrom fail");
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) external nonReentrant updateReward(msg.sender) {
        require(amount > 0 && balances[msg.sender] >= amount, "bad amount");
        totalStaked -= amount;
        balances[msg.sender] -= amount;
        require(axm.transfer(msg.sender, amount), "transfer fail");
        emit Withdrawn(msg.sender, amount);
```

```
    }

    function getReward() external nonReentrant updateReward(msg.sender) {
        uint256 r = rewards[msg.sender];
        require(r > 0, "nothing");
        rewards[msg.sender] = 0;
        require(axm.transfer(msg.sender, r), "transfer fail");
        emit RewardClaimed(msg.sender, r);
    }

    function notifyRewardAmount(uint256 amount, uint256 duration) external nonReentrant
onlyRole(FUNDER_ROLE) updateReward(address(0)) {
        require(amount > 0 && duration > 0, "bad params");
        require(axm.transferFrom(msg.sender, address(this), amount), "fund fail");
        rewardRate = amount / duration;
        lastUpdate = block.timestamp;
        emit RewardAdded(amount, rewardRate, duration);
    }
}
```

9. LiquidityVault.sol LP custody and incentive funding.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract LiquidityVault is AccessControl, ReentrancyGuard {
    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
    bytes32 public constant INCENTIVE_ROLE = keccak256("INCENTIVE_ROLE");

    IERC20 public immutable axm;
```

```
    IERC20 public immutable lpToken;

    address public gauge;
    bool    public lpWithdrawLocked;

    event LPDeposited(address indexed from, uint256 amount);
    event LPWithdrawn(address indexed to, uint256 amount);
    event IncentivesFunded(address indexed to, uint256 amount);
    event GaugeSet(address indexed gauge);
    event LPWithdrawLockSet(bool locked);

    constructor(address admin, address axmToken, address lpTokenAddress) {
        require(admin != address(0) && axmToken != address(0) && lpTokenAddress !=
address(0), "zero addr");
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(MANAGER_ROLE, admin);
        _grantRole(INCENTIVE_ROLE, admin);
        axm = IERC20(axmToken);
        lpToken = IERC20(lpTokenAddress);
    }

    function depositLP(uint256 amount) external nonReentrant onlyRole(MANAGER_ROLE) {
        require(amount > 0, "amount=0");
        require(lpToken.transferFrom(msg.sender, address(this), amount), "transferFrom fail");
        emit LPDeposited(msg.sender, amount);
    }

    function withdrawLP(address to, uint256 amount) external nonReentrant
onlyRole(MANAGER_ROLE) {
        require(!lpWithdrawLocked, "lp locked");
        require(to != address(0) && amount > 0, "bad args");
        require(lpToken.transfer(to, amount), "transfer fail");
        emit LPWithdrawn(to, amount);
```

```
    }

    function setLPWithdrawLocked(bool locked) external onlyRole(MANAGER_ROLE) {
        lpWithdrawLocked = locked;
        emit LPWithdrawLockSet(locked);
    }

    function setGauge(address newGauge) external onlyRole(MANAGER_ROLE) {
        gauge = newGauge;
        emit GaugeSet(newGauge);
    }

    function fundIncentives(uint256 amount, address recipient) external nonReentrant
onlyRole(INCENTIVE_ROLE) {
        require(recipient != address(0) && amount > 0, "bad args");
        require(axm.transferFrom(msg.sender, address(this), amount), "fund fail");
        require(axm.transfer(recipient, amount), "transfer fail");
        emit IncentivesFunded(recipient, amount);
    }

    function rescueToken(address token, address to, uint256 amount) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(to != address(0) && amount > 0, "bad args");
        require(IERC20(token).transfer(to, amount), "rescue fail");
    }
}
```

10. GovernanceCouncil.sol Simple on-chain voting using AXM snapshots.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";
```

```
contract GovernanceCouncil is AccessControl {
    bytes32 public constant PROPOSER_ROLE = keccak256("PROPOSER_ROLE");

    ERC20Votes public immutable axm;
    uint256 public quorumBps;
    uint256 public votingDelayBlocks;
    uint256 public votingPeriodBlocks;

    enum Vote { None, For, Against, Abstain }

    struct Proposal {
        address proposer;
        bytes32 descriptionHash;
        uint256 snapshotBlock;
        uint256 endBlock;
        uint256 forVotes;
        uint256 againstVotes;
        uint256 abstainVotes;
        bool executed;
    }

    uint256 public nextId = 1;
    mapping(uint256 => Proposal) public proposals;
    mapping(uint256 => mapping(address => Vote)) public receipts;

    event ProposalCreated(uint256 id, address proposer, bytes32 descriptionHash, uint256
snapshotBlock, uint256 endBlock);
    event VoteCast(uint256 id, address voter, Vote vote, uint256 weight);
    event MarkExecuted(uint256 id);

    constructor(address admin, address axmToken, uint256 _quorumBps, uint256 _delay, uint256
_period) {
```

```
        require(admin != address(0) && axmToken != address(0), "zero addr");
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(PROPOSER_ROLE, admin);
        axm = ERC20Votes(axmToken);
        quorumBps = _quorumBps;
        votingDelayBlocks = _delay;
        votingPeriodBlocks = _period;
    }

    function setParams(uint256 _quorumBps, uint256 _delay, uint256 _period) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        quorumBps = _quorumBps;
        votingDelayBlocks = _delay;
        votingPeriodBlocks = _period;
    }

    function propose(bytes32 descriptionHash) external onlyRole(PROPOSER_ROLE) returns
(uint256 id) {
        uint256 start = block.number + votingDelayBlocks;
        uint256 end = start + votingPeriodBlocks;
        id = nextId++;
        proposals[id] = Proposal({ proposer: msg.sender, descriptionHash: descriptionHash,
snapshotBlock: start, endBlock: end, forVotes: 0, againstVotes: 0, abstainVotes: 0, executed:
false });
        emit ProposalCreated(id, msg.sender, descriptionHash, start, end);
    }

    function castVote(uint256 id, uint8 voteType) external {
        Proposal storage p = proposals[id];
        require(block.number >= p.snapshotBlock && block.number <= p.endBlock, "not active");
        require(receipts[id][msg.sender] == Vote.None, "voted");
        Vote v = Vote(voteType);
        require(v == Vote.For || v == Vote.Against || v == Vote.Abstain, "bad vote");
        uint256 weight = axm.getPastVotes(msg.sender, p.snapshotBlock);
```

```
        require(weight > 0, "no voting power");
        receipts[id][msg.sender] = v;
        if (v == Vote.For) p.forVotes += weight;
        else if (v == Vote.Against) p.againstVotes += weight;
        else p.abstainVotes += weight;
        emit VoteCast(id, msg.sender, v, weight);
    }

    function quorumReached(uint256 id) public view returns (bool) {
        Proposal memory p = proposals[id];
        uint256 total = axm.getPastTotalSupply(p.snapshotBlock);
        uint256 needed = (total * quorumBps) / 10000;
        return p.forVotes + p.againstVotes + p.abstainVotes >= needed;
    }

    function isSucceeded(uint256 id) public view returns (bool) {
        Proposal memory p = proposals[id];
        if (block.number <= p.endBlock) return false;
        if (!quorumReached(id)) return false;
        return p.forVotes > p.againstVotes;
    }

    function markExecuted(uint256 id) external onlyRole(DEFAULT_ADMIN_ROLE) {
        require(isSucceeded(id), "not succeeded");
        proposals[id].executed = true;
        emit MarkExecuted(id);
    }
}
```

11. OracleRelay.sol Lightweight signed feed cache.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
```

```
import "@openzeppelin/contracts/access/AccessControl.sol";

contract OracleRelay is AccessControl {
    bytes32 public constant ORACLE_ROLE = keccak256("ORACLE_ROLE");

    struct Feed {
        int256 value;
        uint64 updatedAt;
        uint64 ttl;
    }

    mapping(bytes32 => Feed) public feeds;
    event FeedSet(bytes32 indexed key, int256 value, uint64 updatedAt, uint64 ttl);

    constructor(address admin) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(ORACLE_ROLE, admin);
    }

    function setFeed(bytes32 key, int256 value, uint64 ttl) external onlyRole(ORACLE_ROLE) {
        feeds[key] = Feed({ value: value, updatedAt: uint64(block.timestamp), ttl: ttl });
        emit FeedSet(key, value, uint64(block.timestamp), ttl);
    }

    function get(bytes32 key) external view returns (int256 value, bool fresh) {
        Feed memory f = feeds[key];
        if (f.updatedAt == 0) return (0, false);
        fresh = f.ttl == 0 || block.timestamp <= f.updatedAt + f.ttl;
        return (f.value, fresh);
    }
}
```

12. AssetTokenizationRegistry.sol Tokenized asset index for property records.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;


import "@openzeppelin/contracts/access/AccessControl.sol";


contract AssetTokenizationRegistry is AccessControl {

    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");


    enum Status { Inactive, Active, Archived }


    struct Asset {

        string propertyId;

        string location;

        string metadataCid;

        uint256 valuation;

        uint256 backingAmount;

        Status status;

    }


    uint256 public nextId = 1;

    mapping(uint256 => Asset) public assets;


    event Registered(uint256 indexed id, string propertyId, string location);

    event Updated(uint256 indexed id, string metadataCid, uint256 valuation, uint256
backingAmount, Status status);


    constructor(address admin) {

        _grantRole(DEFAULT_ADMIN_ROLE, admin);

        _grantRole(MANAGER_ROLE, admin);

    }


    function registerAsset(string calldata propertyId, string calldata location, string calldata
metadataCid, uint256 valuation, uint256 backingAmount) external onlyRole(MANAGER_ROLE)
returns (uint256 id) {
```

```
        id = nextId++;
        assets[id] = Asset({ propertyId: propertyId, location: location, metadataCid: metadataCid, valuation: valuation, backingAmount: backingAmount, status: Status.Active });
        emit Registered(id, propertyId, location);
        emit Updated(id, metadataCid, valuation, backingAmount, Status.Active);
    }

    function updateAsset(uint256 id, string calldata metadataCid, uint256 valuation, uint256 backingAmount, Status status) external onlyRole(MANAGER_ROLE) {
        Asset storage a = assets[id];
        a.metadataCid = metadataCid;
        a.valuation = valuation;
        a.backingAmount = backingAmount;
        a.status = status;
        emit Updated(id, metadataCid, valuation, backingAmount, status);
    }
}
```

13. TreasuryManager.sol Programmatic treasury payments and sweep.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract TreasuryManager is AccessControl, ReentrancyGuard {
    bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");

    event Pay(address indexed token, address indexed to, uint256 amount, bytes32 refId);
    event Sweep(address indexed token, address indexed to, uint256 amount);

    constructor(address admin) {
```

```
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(OPERATOR_ROLE, admin);
    }

    function pay(address token, address to, uint256 amount, bytes32 refId) external
nonReentrant onlyRole(OPERATOR_ROLE) {
        require(token != address(0) && to != address(0) && amount > 0, "bad args");
        require(IERC20(token).transfer(to, amount), "transfer fail");
        emit Pay(token, to, amount, refId);
    }

    function sweep(address token, address to, uint256 amount) external nonReentrant
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(token != address(0) && to != address(0) && amount > 0, "bad args");
        require(IERC20(token).transfer(to, amount), "transfer fail");
        emit Sweep(token, to, amount);
    }
}
```

14. AXIOMRevenueRouter.sol Split AXM inflows to protocol sinks.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract AXIOMRevenueRouter is AccessControl {
    IERC20 public immutable axm;

    address public raf;
    address public depinTreasury;
    address public liquidity;
    address public treasury;
```

```
    uint16 public bpRaf;

    uint16 public bpDepin;

    uint16 public bpLiq;

    uint16 public bpTsy;


    event ConfigUpdated(address raf, address depinT, address liq, address tsy, uint16 rafBp,
uint16 depinBp, uint16 liqBp, uint16 tsyBp);

    event Routed(uint256 amount, uint256 toRaf, uint256 toDepin, uint256 toLiq, uint256 toTsy);


    constructor(address admin, address _axm, address _raf, address _depinTreasury, address
_liquidity, address _treasury,

            uint16 _rafBp, uint16 _depinBp, uint16 _liqBp, uint16 _tsyBp) {

        _grantRole(DEFAULT_ADMIN_ROLE, admin);

        axm = IERC20(_axm);

        _setConfig(_raf, _depinTreasury, _liquidity, _treasury, _rafBp, _depinBp, _liqBp, _tsyBp);

    }


    function setConfig(address _raf, address _depinTreasury, address _liquidity, address
_treasury,

                uint16 _rafBp, uint16 _depinBp, uint16 _liqBp, uint16 _tsyBp) external
onlyRole(DEFAULT_ADMIN_ROLE) {

        _setConfig(_raf, _depinTreasury, _liquidity, _treasury, _rafBp, _depinBp, _liqBp, _tsyBp);

    }


    function _setConfig(address _raf, address _depinTreasury, address _liquidity, address
_treasury,

                uint16 _rafBp, uint16 _depinBp, uint16 _liqBp, uint16 _tsyBp) internal {

        require(_raf != address(0) && _depinTreasury != address(0) && _liquidity != address(0) &&
_treasury != address(0), "zero addr");

        require(uint32(_rafBp) + _depinBp + _liqBp + _tsyBp == 10_000, "bps!=100");

        raf = _raf; depinTreasury = _depinTreasury; liquidity = _liquidity; treasury = _treasury;

        bpRaf = _rafBp; bpDepin = _depinBp; bpLiq = _liqBp; bpTsy = _tsyBp;

        emit ConfigUpdated(raf, depinTreasury, liquidity, treasury, bpRaf, bpDepin, bpLiq, bpTsy);

    }
```

```
    function route() external {
        uint256 bal = axm.balanceOf(address(this));
        require(bal > 0, "no funds");
        uint256 toRaf   = (bal * bpRaf)   / 10_000;
        uint256 toDepin = (bal * bpDepin) / 10_000;
        uint256 toLiq   = (bal * bpLiq)   / 10_000;
        uint256 toTsy   = bal - toRaf - toDepin - toLiq;
        require(axm.transfer(raf, toRaf) && axm.transfer(depinTreasury, toDepin) &&
axm.transfer(liquidity, toLiq) && axm.transfer(treasury, toTsy), "transfer fail");
        emit Routed(bal, toRaf, toDepin, toLiq, toTsy);
    }
}
```

15. RealEstateAcquisitionFund.sol Simple share points and AXM distribution.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract RealEstateAcquisitionFund is AccessControl, ReentrancyGuard {
    bytes32 public constant DISTRIBUTOR_ROLE = keccak256("DISTRIBUTOR_ROLE");
    IERC20 public immutable axm;

    mapping(address => uint256) public shares;
    uint256 public totalShares;

    event Funded(address indexed from, uint256 amount);
    event SharesSet(address indexed user, uint256 oldVal, uint256 newVal);
    event Distributed(uint256 total, uint256 recipients);
```

```
    constructor(address admin, address _axm) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(DISTRIBUTOR_ROLE, admin);
        axm = IERC20(_axm);
    }

    function topUp(uint256 amount) external {
        require(amount > 0, "amount=0");
        require(axm.transferFrom(msg.sender, address(this), amount), "transferFrom fail");
        emit Funded(msg.sender, amount);
    }

    function setShares(address user, uint256 newVal) external onlyRole(DISTRIBUTOR_ROLE) {
        uint256 old = shares[user];
        totalShares = totalShares + newVal - old;
        shares[user] = newVal;
        emit SharesSet(user, old, newVal);
    }

    function batchSetShares(address[] calldata users, uint256[] calldata vals) external
onlyRole(DISTRIBUTOR_ROLE) {
        require(users.length == vals.length, "len mismatch");
        for (uint256 i = 0; i < users.length; i++) {
            uint256 old = shares[users[i]];
            totalShares = totalShares + vals[i] - old;
            shares[users[i]] = vals[i];
            emit SharesSet(users[i], old, vals[i]);
        }
    }

    function distribute(address[] calldata recipients) external nonReentrant
onlyRole(DISTRIBUTOR_ROLE) {
        uint256 bal = axm.balanceOf(address(this));
```

```
        require(bal > 0 && totalShares > 0 && recipients.length > 0, "nothing");
        uint256 sent = 0;
        for (uint256 i = 0; i < recipients.length; i++) {
            uint256 s = shares[recipients[i]];
            if (s == 0) continue;
            uint256 amt = (bal * s) / totalShares;
            if (amt > 0) {
                require(axm.transfer(recipients[i], amt), "transfer fail");
                sent += amt;
            }
        }
        emit Distributed(sent, recipients.length);
    }
}
```

16. AXIOMLaunchpad.sol AXM-denominated presale escrow with soft and hard caps and refund logic.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract AXIOMLaunchpad is AccessControl, ReentrancyGuard {
    bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
    IERC20 public immutable axm;

    uint64  public start;
    uint64  public end;
    uint256 public softCap;
    uint256 public hardCap;
    uint256 public price;
```

```
    bool    public cancelled;
    uint256 public raised;

    mapping(address => uint256) public contributed;
    mapping(address => uint256) public allocation;

    event ConfigSet(uint64 start, uint64 end, uint256 softCap, uint256 hardCap, uint256 price);
    event Contributed(address indexed user, uint256 axmAmount, uint256 unitsOut);
    event Cancelled();
    event Withdrawn(address indexed to, uint256 amount);
    event Refunded(address indexed user, uint256 amount);

    constructor(address admin, address axmToken, uint64 _start, uint64 _end, uint256 _softCap,
uint256 _hardCap, uint256 _price) {
        require(admin != address(0) && axmToken != address(0), "zero addr");
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(MANAGER_ROLE, admin);
        axm = IERC20(axmToken);
        setConfig(_start, _end, _softCap, _hardCap, _price);
    }

    function setConfig(uint64 _start, uint64 _end, uint256 _softCap, uint256 _hardCap, uint256
_price) public onlyRole(MANAGER_ROLE) {
        require(_start < _end && _price > 0 && _hardCap >= _softCap, "bad params");
        start = _start; end = _end; softCap = _softCap; hardCap = _hardCap; price = _price;
        emit ConfigSet(start, end, softCap, hardCap, price);
    }

    function contribute(uint256 axmAmount) external nonReentrant {
        require(!cancelled, "cancelled");
        require(block.timestamp >= start && block.timestamp <= end, "not active");
        require(axmAmount > 0 && raised + axmAmount <= hardCap, "cap reached");
```

```
        require(axm.transferFrom(msg.sender, address(this), axmAmount), "transferFrom fail");
        raised += axmAmount;
        contributed[msg.sender] += axmAmount;
        uint256 units = (axmAmount * price) / 1e18;
        allocation[msg.sender] += units;
        emit Contributed(msg.sender, axmAmount, units);
    }

    function cancel() external onlyRole(MANAGER_ROLE) {
        cancelled = true;
        emit Cancelled();
    }

    function withdrawProceeds(address to) external onlyRole(MANAGER_ROLE) nonReentrant {
        require(!cancelled, "cancelled");
        require(block.timestamp > end && raised >= softCap, "not finalized");
        uint256 bal = axm.balanceOf(address(this));
        require(axm.transfer(to, bal), "transfer fail");
        emit Withdrawn(to, bal);
    }

    function refund() external nonReentrant {
        require(cancelled || (block.timestamp > end && raised < softCap), "no refund");
        uint256 amt = contributed[msg.sender];
        require(amt > 0, "none");
        contributed[msg.sender] = 0;
        allocation[msg.sender] = 0;
        require(axm.transfer(msg.sender, amt), "transfer fail");
        emit Refunded(msg.sender, amt);
    }
}
```

17. CrossChainBridgeAdapter.sol Canonical locker for bridging.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;


import "@openzeppelin/contracts/access/AccessControl.sol";

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";


contract CrossChainBridgeAdapter is AccessControl, ReentrancyGuard {

    bytes32 public constant RELAYER_ROLE = keccak256("RELAYER_ROLE");

    IERC20 public immutable axm;


    event Locked(address indexed from, bytes32 indexed dstChain, bytes32 indexed dstAddress,
uint256 amount, bytes32 lockId);

    event Unlocked(address indexed to, uint256 amount, bytes32 srcChain, bytes32 burnTx);


    mapping(bytes32 => bool) public processed;


    constructor(address admin, address axmToken) {

        _grantRole(DEFAULT_ADMIN_ROLE, admin);

        _grantRole(RELAYER_ROLE, admin);

        axm = IERC20(axmToken);

    }


    function lock(bytes32 dstChain, bytes32 dstAddress, uint256 amount, bytes32 lockId)
external nonReentrant {

        require(amount > 0, "amount=0");

        require(!processed[lockId], "dup");

        processed[lockId] = true;

        require(axm.transferFrom(msg.sender, address(this), amount), "transferFrom fail");

        emit Locked(msg.sender, dstChain, dstAddress, amount, lockId);

    }


    function unlock(address to, uint256 amount, bytes32 srcChain, bytes32 burnTx) external
nonReentrant onlyRole(RELAYER_ROLE) {
```

```
        require(to != address(0) && amount > 0, "bad args");
        require(!processed[burnTx], "done");
        processed[burnTx] = true;
        require(axm.transfer(to, amount), "transfer fail");
        emit Unlocked(to, amount, srcChain, burnTx);
    }
}
```

18. AXIOMAnalyticsHub.sol CID and digest publication per node per epoch.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";

contract AXIOMAnalyticsHub is AccessControl {
    bytes32 public constant PUBLISHER_ROLE = keccak256("PUBLISHER_ROLE");
    mapping(uint256 => mapping(uint256 => bytes32)) public reportDigest;
    mapping(uint256 => mapping(uint256 => bytes32)) public reportCid;
    event Published(uint256 indexed nodeId, uint256 indexed epoch, bytes32 digest, bytes32
cid);

    constructor(address admin) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(PUBLISHER_ROLE, admin);
    }

    function publish(uint256 nodeId, uint256 epoch, bytes32 digest, bytes32 cid) external
onlyRole(PUBLISHER_ROLE) {
        require(nodeId != 0 && epoch != 0 && digest != bytes32(0), "bad args");
        reportDigest[nodeId][epoch] = digest;
        if (cid != bytes32(0)) reportCid[nodeId][epoch] = cid;
        emit Published(nodeId, epoch, digest, cid);
    }
```

```
    function get(uint256 nodeId, uint256 epoch) external view returns (bytes32 digest, bytes32
cid) {
        return (reportDigest[nodeId][epoch], reportCid[nodeId][epoch]);
    }
}
```

19. Liquidity note If you plan to use an LP token address, provide it during LiquidityVault deployment. Otherwise, deploy LiquidityVault with a known LP token later.
20. Optional property share modules If needed later, we can add PropertyShareToken and LeaseManager as separate tasks. Omit for now to keep the suite lean and low risk.

Deployment utilities Create scripts/utils.js

```
const fs = require("fs");
const path = require("path");
const ADDR_PATH = path.join(__dirname, "..", "addresses.json");

function saveAddress(key, value) {
  let data = {};
  if (fs.existsSync(ADDR_PATH)) {
    data = JSON.parse(fs.readFileSync(ADDR_PATH, "utf8"));
  }
  data[key] = value;
  fs.writeFileSync(ADDR_PATH, JSON.stringify(data, null, 2));
  console.log("Saved", key, "=", value);
}

function getAddress(key) {
  const data = JSON.parse(fs.readFileSync(ADDR_PATH, "utf8"));
  return data[key];
}

module.exports = { saveAddress, getAddress, ADDR_PATH };
```

Deployment scripts Create the following under scripts/. Each prints and saves to addresses.json, then does minimal role grants.

00_deploy_AXM.js

```
const { saveAddress } = require("./utils");


module.exports = async function() {
  const [deployer] = await ethers.getSigners();
  const vault = process.env.DISTRIBUTION_VAULT;
  if (!vault) throw new Error("DISTRIBUTION_VAULT missing");
  const AXM = await ethers.getContractFactory("AXM");
  const axm = await AXM.deploy(vault);
  await axm.deployed();
  console.log("AXM:", axm.address);
  saveAddress("AXM", axm.address);
};
module.exports.tags = ["AXM"];
```

01_deploy_EmissionsVault.js

```
const { saveAddress, getAddress } = require("./utils");


module.exports = async function() {
  const [deployer] = await ethers.getSigners();
  const axm = getAddress("AXM");
  const beneficiary = process.env.ADMIN;
  const start = Math.floor(Date.now() / 1000);
  const duration = 60n * 60n * 24n * 365n; // one year
  const EmissionsVault = await ethers.getContractFactory("EmissionsVault");
  const vault = await EmissionsVault.deploy(axm, beneficiary, start, duration);
  await vault.deployed();
  console.log("EmissionsVault:", vault.address);
  saveAddress("EmissionsVault", vault.address);
};
module.exports.tags = ["EmissionsVault"];
```

```
module.exports.dependencies = ["AXM"];
```

02_deploy_DID_and_Compliance.js

```
const { saveAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const kyc = process.env.KYC_SCHEMA;
  const acc = process.env.ACCREDITED_SCHEMA;
  const nonus = process.env.NON_US_SCHEMA;
  const DID = await ethers.getContractFactory("DIDRegistry");
  const did = await DID.deploy(admin);
  await did.deployed();
  console.log("DIDRegistry:", did.address);
  saveAddress("DIDRegistry", did.address);

  const Compliance = await ethers.getContractFactory("ComplianceAdapter");
  const comp = await Compliance.deploy(admin, did.address, kyc, acc, nonus);
  await comp.deployed();
  console.log("ComplianceAdapter:", comp.address);
  saveAddress("ComplianceAdapter", comp.address);
};
module.exports.tags = ["DID","Compliance"];
```

03_deploy_DePIN_core.js

```
const { saveAddress, getAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const axm = getAddress("AXM");
  const did = getAddress("DIDRegistry");
  const licenseSchema = process.env.LICENSE_SCHEMA;
  const kycSchema = process.env.KYC_SCHEMA;
```

```
  const NodeLicense = await ethers.getContractFactory("NodeLicenseNFT");
  const nodeLicense = await NodeLicense.deploy(admin);
  await nodeLicense.deployed();
  saveAddress("NodeLicenseNFT", nodeLicense.address);

  const Registry = await ethers.getContractFactory("DePINNodeRegistry");
  const reg = await Registry.deploy(admin, did, licenseSchema);
  await reg.deployed();
  saveAddress("DePINNodeRegistry", reg.address);

  const Router = await ethers.getContractFactory("DePINRewardRouter");
  const router = await Router.deploy(admin, axm, reg.address, did, kycSchema);
  await router.deployed();
  saveAddress("DePINRewardRouter", router.address);

  const Hub = await ethers.getContractFactory("AXIOMAnalyticsHub");
  const hub = await Hub.deploy(admin);
  await hub.deployed();
  saveAddress("AXIOMAnalyticsHub", hub.address);

  console.log("DePIN core deployed");
};
module.exports.tags = ["DePIN"];
module.exports.dependencies = ["AXM","DID","Compliance"];
```

04_deploy_Liquidity_and_Staking.js

```
const { saveAddress, getAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const axm = getAddress("AXM");
```

```
  const lpToken = "0x0000000000000000000000000000000000000001"; // replace with actual
LP if available

  const LV = await ethers.getContractFactory("LiquidityVault");
  const lv = await LV.deploy(admin, axm, lpToken);
  await lv.deployed();
  saveAddress("LiquidityVault", lv.address);

  const Pool = await ethers.getContractFactory("AXMStakingPool");
  const pool = await Pool.deploy(admin, axm);
  await pool.deployed();
  saveAddress("AXMStakingPool", pool.address);

  console.log("LiquidityVault:", lv.address);
  console.log("AXMStakingPool:", pool.address);
};
module.exports.tags = ["Liquidity","Staking"];
module.exports.dependencies = ["AXM"];
```

05_deploy_Oracle_and_Governance.js

```
const { saveAddress, getAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const axm = getAddress("AXM");

  const Oracle = await ethers.getContractFactory("OracleRelay");
  const oracle = await Oracle.deploy(admin);
  await oracle.deployed();
  saveAddress("OracleRelay", oracle.address);

  const Gov = await ethers.getContractFactory("GovernanceCouncil");
```

```
  const gov = await Gov.deploy(admin, axm, 1000, 5, 20); // quorum 10 percent, delay 5 blocks, period 20 blocks

  await gov.deployed();

  saveAddress("GovernanceCouncil", gov.address);


  console.log("OracleRelay:", oracle.address);

  console.log("GovernanceCouncil:", gov.address);

};

module.exports.tags = ["Oracle","Governance"];

module.exports.dependencies = ["AXM"];
```

06_deploy_Assets_and_Treasury.js

```
const { saveAddress, getAddress } = require("./utils");


module.exports = async function() {

  const admin = process.env.ADMIN;

  const axm = getAddress("AXM");


  const ATR = await ethers.getContractFactory("AssetTokenizationRegistry");

  const atr = await ATR.deploy(admin);

  await atr.deployed();

  saveAddress("AssetTokenizationRegistry", atr.address);


  const TM = await ethers.getContractFactory("TreasuryManager");

  const tm = await TM.deploy(admin);

  await tm.deployed();

  saveAddress("TreasuryManager", tm.address);


  const RAF = await ethers.getContractFactory("RealEstateAcquisitionFund");

  const raf = await RAF.deploy(admin, axm);

  await raf.deployed();

  saveAddress("RealEstateAcquisitionFund", raf.address);
```

```
  console.log("AssetTokenizationRegistry:", atr.address);
  console.log("TreasuryManager:", tm.address);
  console.log("RealEstateAcquisitionFund:", raf.address);
};
module.exports.tags = ["Assets","Treasury","RAF"];
module.exports.dependencies = ["AXM"];
```

07_deploy_Revenue_and_Launchpad.js

```
const { saveAddress, getAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const axm = getAddress("AXM");

  const rafSink   = process.env.RAF_SINK;
  const depinSink = process.env.DEPIN_TREASURY_SINK;
  const liqSink   = process.env.LIQUIDITY_SINK;
  const tsySink   = process.env.TREASURY_SINK;

  const RR = await ethers.getContractFactory("AXIOMRevenueRouter");
  const rr = await RR.deploy(admin, axm, rafSink, depinSink, liqSink, tsySink, 3000, 3000, 2000,
2000);
  await rr.deployed();
  saveAddress("AXIOMRevenueRouter", rr.address);

  const now = Math.floor(Date.now() / 1000);
  const start = now + 3600;
  const end = start + 14 * 24 * 3600;
  const softCap = ethers.parseUnits("1000000", 18);
  const hardCap = ethers.parseUnits("5000000", 18);
  const price   = ethers.parseUnits("1000", 18); // units per 1 AXM

  const LP = await ethers.getContractFactory("AXIOMLaunchpad");
```

```
  const lp = await LP.deploy(admin, axm, start, end, softCap, hardCap, price);
  await lp.deployed();
  saveAddress("AXIOMLaunchpad", lp.address);

  console.log("AXIOMRevenueRouter:", rr.address);
  console.log("AXIOMLaunchpad:", lp.address);
};
module.exports.tags = ["Revenue","Launchpad"];
module.exports.dependencies = ["AXM"];

08_deploy_Bridge.js
const { saveAddress, getAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const axm = getAddress("AXM");
  const Bridge = await ethers.getContractFactory("CrossChainBridgeAdapter");
  const bridge = await Bridge.deploy(admin, axm);
  await bridge.deployed();
  saveAddress("CrossChainBridgeAdapter", bridge.address);
  console.log("CrossChainBridgeAdapter:", bridge.address);
};
module.exports.tags = ["Bridge"];
module.exports.dependencies = ["AXM"];
```

Role grants helper (optional) Add scripts/grant_roles_example.js showing how to grant roles you need after deployment.

```
const { getAddress } = require("./utils");

module.exports = async function() {
  const admin = process.env.ADMIN;
  const [caller] = await ethers.getSigners();
  console.log("Granting roles from", caller.address);
```

```
  const did = await ethers.getContractAt("DIDRegistry", getAddress("DIDRegistry"));

  const comp = await ethers.getContractAt("ComplianceAdapter",
getAddress("ComplianceAdapter"));

  const reg = await ethers.getContractAt("DePINNodeRegistry",
getAddress("DePINNodeRegistry"));

  const router = await ethers.getContractAt("DePINRewardRouter",
getAddress("DePINRewardRouter"));

  const hub = await ethers.getContractAt("AXIOMAnalyticsHub",
getAddress("AXIOMAnalyticsHub"));

  const license = await ethers.getContractAt("NodeLicenseNFT",
getAddress("NodeLicenseNFT"));

  const pool = await ethers.getContractAt("AXMStakingPool", getAddress("AXMStakingPool"));

  const lv = await ethers.getContractAt("LiquidityVault", getAddress("LiquidityVault"));

  const oracle = await ethers.getContractAt("OracleRelay", getAddress("OracleRelay"));


  const DEFAULT_ADMIN_ROLE = await did.DEFAULT_ADMIN_ROLE();


  // Example: ensure deployer has admin where needed

  await did.grantRole(DEFAULT_ADMIN_ROLE, admin);

  await license.grantRole(await license.ISSUER_ROLE(), admin);

  await pool.grantRole(await pool.FUNDER_ROLE(), admin);

  await lv.grantRole(await lv.MANAGER_ROLE(), admin);


  console.log("Roles granted");

};

module.exports.tags = ["GrantRoles"];


Package.json If missing, create a minimal package.json:

{

  "name": "axiom-peaq-suite",

  "version": "1.0.0",

  "license": "MIT",

  "scripts": {

    "compile": "hardhat compile",
```

```
    "deploy:axm": "hardhat run scripts/00_deploy_AXM.js --network peaq",
    "deploy:vest": "hardhat run scripts/01_deploy_EmissionsVault.js --network peaq",
    "deploy:did": "hardhat run scripts/02_deploy_DID_and_Compliance.js --network peaq",
    "deploy:depin": "hardhat run scripts/03_deploy_DePIN_core.js --network peaq",
    "deploy:liq": "hardhat run scripts/04_deploy_Liquidity_and_Staking.js --network peaq",
    "deploy:gov": "hardhat run scripts/05_deploy_Oracle_and_Governance.js --network peaq",
    "deploy:assets": "hardhat run scripts/06_deploy_Assets_and_Treasury.js --network peaq",
    "deploy:router": "hardhat run scripts/07_deploy_Revenue_and_Launchpad.js --network
peaq",
    "deploy:bridge": "hardhat run scripts/08_deploy_Bridge.js --network peaq",
    "roles": "hardhat run scripts/grant_roles_example.js --network peaq"
  },
  "devDependencies": {
    "@nomicfoundation/hardhat-toolbox": "^5.0.0",
    "dotenv": "^16.4.5",
    "hardhat": "^2.22.5"
  },
  "dependencies": {
    "@openzeppelin/contracts": "^5.0.2"
  }
}
```

Run sequence

1. Fill .env with correct values. For first deployment, you can set DISTRIBUTION_VAULT to your ADMIN to receive initial AXM supply.
2. Compile npm run compile
3. Deploy in order npm run deploy:axm npm run deploy:vest npm run deploy:did npm run deploy:depin npm run deploy:liq npm run deploy:gov npm run deploy:assets npm run deploy:router npm run deploy:bridge
4. Grant roles if needed npm run roles
5. Confirm addresses.json exists at project root with all contract addresses recorded.

Quality checklist

1. All contracts compile under Solidity 0.8.20 with optimizer enabled.
2. No minting beyond AXM constructor mint to distribution vault.
3. All access controls are constructor-wired to ADMIN.
4. No upgradeable proxies are used to keep risk low.
5. Launchpad supports presale in AXM with soft and hard caps and refund guarantees.

Deliverables

1. New contracts in contracts/
2. New deploy scripts in scripts/
3. hardhat.config.js and package.json as specified
4. .env template keys populated
5. addresses.json written during deploy

## End of Replit AI prompt