1) EmissionsVault.sol — linear vesting/funding vault

- Holds AXM and releases linearly over time to a beneficiary or to any module you fund.
- No owner withdrawals beyond the linear schedule; supports multiple “funding lanes” if you want.

```
// 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;   // typically your ops multisig or RewardsDistributor
    uint64  public immutable start;         // vesting start timestamp
    uint64  public immutable duration;      // vesting duration in seconds
    uint256 public released;                // AXM already 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 _vestingEarned() - released;
    }

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

    function _vestingEarned() 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;
    }
}
```

2) DIDRegistry.sol — verifiable-credential hashes (no PII)

- Stores credential hashes by schema for an address, with issuer, issuedAt, expiry, and status.
- Schemas you'll use: `KYC_BASIC_V1`, `ACCREDITED_V1`, `NON_US_V1`, `DENET_NODE_LICENSE_V1`.

```
// 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;      // hash of VC payload
        address issuer;    // on-chain issuer account
        uint64  issuedAt;
        uint64  expiry;
        bool    revoked;
    }

    // subject => schema => credential
    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, "already 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 expiry");
        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);
    }
}
```

3) DePINNodeRegistry.sol — bind nodes to operators & licenses

- Registers a node to an operator wallet.
- Links to DIDRegistry to require a valid DeNet Node License credential.

```
// 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; // e.g.,
keccak256("DENET_NODE_LICENSE_V1")

    enum Status { Inactive, Active }

    struct Node {
        address operator;
        bytes32 deviceDid;      // hashed device DID string
        bytes32 licenseHash;    // hashed license VC payload
        bytes32 metadataCid;    // IPFS/IPNS CID hash
        uint96  weight;         // relative reward 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");
        // credential must exist and not be revoked/expired
        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];
    }
}
```

4) DePINRewardRouter.sol — epoch settlement & AXM payouts

- Oracle posts per-epoch scores per node.
- Router accrues rewards and lets operators claim AXM from a funded balance.

```
// 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; // optional gating

  // epoch => settled?
  mapping(uint256 => bool) public epochSettled;
  // node => accrued AXM
  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; // set to 0x0 to disable KYC gate
  }

  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);
  }

  // Oracle provides nodeIds and scores for the epoch; router converts to payouts by proportion
of total weight*score.
  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 funds");

        // compute weighted sum
        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");

        // accrue
        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);
    }
}
```

5) AXIOMRevenueRouter.sol — split protocol revenues

- Accepts AXM and splits to configured sinks (RAF, Rewards Treasury, Liquidity, Treasury).
- Percentages are in basis points and sum to 10,000.

```
// 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;          // RealEstateAcquisitionFund
    address public depinTreasury; // DePINRewardRouter funding wallet
    address public liquidity;    // LP ops wallet
    address public treasury;     // general 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);
    }
}
```

6) RealEstateAcquisitionFund.sol — investor distribution (AXM only)

- Holds AXM and distributes to investors according to tiers or unit balances you maintain off-chain or by staking contract.

```
// 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;

  // Investor accounting kept minimal; plug your own staking or share token externally.
  mapping(address => uint256) public shares;  // abstract "points"
  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);
    }
}
```

That's the "PEAQ-first" minimal, powerful suite: fixed AXM, emissions, DID, node registry, rewards, revenue router, and RAF. Next natural additions if you want me to keep going:

- Minimal NFT marketplace (AXM payments), or property share ERC20/NFT.
- Timelock/Multisig guard contract scaffolding.
- Hardhat/Foundry deploy scripts and addresses.json template so everything is wired with constructors and pinned events from day one.

A)

Initialize PEAQ Contract Suite

Create a new Solidity project structure with the following files inside the contracts/ folder:

AXM.sol

EmissionsVault.sol

DIDRegistry.sol

DePINNodeRegistry.sol

DePINRewardRouter.sol

AXIOMRevenueRouter.sol

RealEstateAcquisitionFund.sol

Use the Solidity code provided in our chat for each file exactly.

Initialize the project for Hardhat (or Foundry if preferred), install OpenZeppelin contracts (npm install @openzeppelin/contracts), and verify that each contract compiles successfully on Solidity ^0.8.20.

After that, create deployment scripts in a scripts/ folder:

1. 00_deploy_AXM.js – deploys AXM with constructor param distributionVault.

2. 01_deploy_EmissionsVault.js – deploys vault, linking to AXM address.

3. 02_deploy_DIDRegistry.js – deploys registry with admin param.

4. 03_deploy_DePIN.js – deploys node registry and reward router.

5. 04_deploy_RevenueAndRAF.js – deploys revenue router and RAF.

Configure hardhat.config.js to include Peaq’s EVM RPC and chain ID. Chain RPC: https://evm.peaq.network

Chain ID: 3338

Verify compilation output and generate the artifacts.

Once Replit AI executes that prompt, your repo will have a fully organized, compilable contract suite.

Then, when ready, you'll:

1. Import your deployer wallet's private key in .env.

2. Run npx hardhat run scripts/00_deploy_AXM.js --network peaq.

3. Proceed down the list to deploy all modules on Peaq testnet or mainnet.

Here are ready-to-use Hardhat deploy scripts and config for a Peaq launch. Paste these into your Replit project and run in order. No owners, no surprises, every constructor wired up front.

hardhat.config.js

```
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: 200 } }

  },

  networks: {

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

package.json

```
{
  "name": "axiom-peaq",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "compile": "hardhat compile",
    "deploy:axm": "hardhat run scripts/00_deploy_AXM.js --network peaq",
    "deploy:vault": "hardhat run scripts/01_deploy_EmissionsVault.js --network peaq",
    "deploy:did": "hardhat run scripts/02_deploy_DIDRegistry.js --network peaq",
    "deploy:depin": "hardhat run scripts/03_deploy_DePIN.js --network peaq",
    "deploy:revenue": "hardhat run scripts/04_deploy_RevenueAndRAF.js --network peaq"
  },
  "devDependencies": {
    "@nomicfoundation/hardhat-toolbox": "^5.0.0",
    "dotenv": "^16.4.5",
    "hardhat": "^2.22.5"
  }
}
```

.env.sample

```
PEAQ_RPC=https://evm.peaq.network
DEPLOYER_PK=0xyour_private_key_without_quotes
DISTRIBUTION_VAULT=0xYourMultisigVault
```

```
ADMIN=0xYourAdminMultisig

RAF_DISTRIBUTOR=0xYourRAFDistributorRoleHolder


SCHEMA_LICENSE=0x44454e45545f4e4f44455f4c4943454e53455f5631

SCHEMA_KYC=0x4b59435f42415349435f5631


EMISSIONS_START=1731024000

EMISSIONS_DURATION=31536000


RAF_BP=2000

DEPIN_BP=4000

LIQ_BP=2000

TSY_BP=2000


DEPIN_TREASURY=0xYourDePINFundingWallet

LIQ_WALLET=0xYourLiquidityWallet

TSY_WALLET=0xYourTreasuryWallet
```

scripts/helpers/saveAddress.js

```
const fs = require("fs");

const path = require("path");


function saveAddress(network, name, address, extras = {}) {

  const p = path.join(__dirname, "..", "..", "config", "addresses.json");

  let json = {};

  if (fs.existsSync(p)) json = JSON.parse(fs.readFileSync(p, "utf8"));

  if (!json[network]) json[network] = {};

  json[network][name] = { address, ...extras, updatedAt: new Date().toISOString() };

  fs.mkdirSync(path.dirname(p), { recursive: true });

  fs.writeFileSync(p, JSON.stringify(json, null, 2));

  console.log(`Saved ${name} at ${address}`);

}
```

```
module.exports = { saveAddress };
```

scripts/00_deploy_AXM.js

```
const { ethers, network } = require("hardhat");
const { saveAddress } = require("./helpers/saveAddress");

async function main() {
  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.waitForDeployment();
  const addr = await axm.getAddress();
  console.log(`AXM deployed at ${addr}`);
  saveAddress(network.name, "AXM", addr, { vault });
}
main().catch((e) => { console.error(e); process.exit(1); });
```

scripts/01_deploy_EmissionsVault.js

```
const { ethers, network } = require("hardhat");
const { saveAddress } = require("./helpers/saveAddress");
const fs = require("fs");

function getAddr(name) {
  const j = JSON.parse(fs.readFileSync("config/addresses.json","utf8"));
  return j[network.name][name].address;
}

async function main() {
  const axm = getAddr("AXM");
  const beneficiary = process.env.ADMIN;
  const start = BigInt(process.env.EMISSIONS_START || "0");
  const duration = BigInt(process.env.EMISSIONS_DURATION || "31536000");
  if (!beneficiary) throw new Error("ADMIN missing");

  const EmissionsVault = await ethers.getContractFactory("EmissionsVault");
  const vault = await EmissionsVault.deploy(axm, beneficiary, start, duration);
  await vault.waitForDeployment();
  const addr = await vault.getAddress();
  console.log(`EmissionsVault at ${addr}`);
  saveAddress(network.name, "EmissionsVault", addr, { start: start.toString(), duration:
duration.toString(), beneficiary });
}
main().catch((e)=>{ console.error(e); process.exit(1); });
```

scripts/02_deploy_DIDRegistry.js

```
const { ethers, network } = require("hardhat");
const { saveAddress } = require("./helpers/saveAddress");

async function main() {
  const admin = process.env.ADMIN;
  if (!admin) throw new Error("ADMIN missing");

  const DIDRegistry = await ethers.getContractFactory("DIDRegistry");
  const did = await DIDRegistry.deploy(admin);
  await did.waitForDeployment();
  const addr = await did.getAddress();
  console.log(`DIDRegistry at ${addr}`);
  saveAddress(network.name, "DIDRegistry", addr);

  // Optional role grants can be done here with did.grantRole if you already know issuer
addresses
}

main().catch((e)=>{ console.error(e); process.exit(1); });
```

scripts/03_deploy_DePIN.js

```
const { ethers, network } = require("hardhat");
const { saveAddress } = require("./helpers/saveAddress");
const fs = require("fs");

function getAddr(name) {
  const j = JSON.parse(fs.readFileSync("config/addresses.json","utf8"));
  return j[network.name][name].address;
}

async function main() {
  const admin = process.env.ADMIN;
  const axm = getAddr("AXM");
  const did = getAddr("DIDRegistry");

  const schemaLicense = process.env.SCHEMA_LICENSE;
  const schemaKyc = process.env.SCHEMA_KYC || "0x";
  if (!schemaLicense) throw new Error("SCHEMA_LICENSE missing");

  const DePINNodeRegistry = await ethers.getContractFactory("DePINNodeRegistry");
  const nodeReg = await DePINNodeRegistry.deploy(admin, did, schemaLicense);
  await nodeReg.waitForDeployment();
  const nodeRegAddr = await nodeReg.getAddress();
  console.log(`DePINNodeRegistry at ${nodeRegAddr}`);
  saveAddress(network.name, "DePINNodeRegistry", nodeRegAddr, { schemaLicense });

  const DePINRewardRouter = await ethers.getContractFactory("DePINRewardRouter");
  const router = await DePINRewardRouter.deploy(admin, axm, nodeRegAddr, did, schemaKyc);
  await router.waitForDeployment();
  const routerAddr = await router.getAddress();

  console.log(`DePINRewardRouter at ${routerAddr}`);

  saveAddress(network.name, "DePINRewardRouter", routerAddr, { schemaKyc });

}

main().catch((e)=>{ console.error(e); process.exit(1); });
```

scripts/04_deploy_RevenueAndRAF.js

```
const { ethers, network } = require("hardhat");

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

const fs = require("fs");


function getAddr(name) {

  const j = JSON.parse(fs.readFileSync("config/addresses.json","utf8"));

  return j[network.name][name].address;

}


async function main() {

  const admin = process.env.ADMIN;

  const axm = getAddr("AXM");


  const rafDistributor = process.env.RAF_DISTRIBUTOR;

  if (!rafDistributor) throw new Error("RAF_DISTRIBUTOR missing");


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

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

  await raf.waitForDeployment();

  const rafAddr = await raf.getAddress();

  console.log(`RAF at ${rafAddr}`);

  saveAddress(network.name, "RealEstateAcquisitionFund", rafAddr);


  // grant distributor role
  const DISTRIBUTOR_ROLE = await raf.DISTRIBUTOR_ROLE();
  await (await raf.grantRole(DISTRIBUTOR_ROLE, rafDistributor)).wait();
  console.log(`Granted DISTRIBUTOR_ROLE to ${rafDistributor}`);

  const depinTreasury = process.env.DEPIN_TREASURY;
  const liq = process.env.LIQ_WALLET;
  const tsy = process.env.TSY_WALLET;

  const bpRaf = Number(process.env.RAF_BP || 2000);
  const bpDepin = Number(process.env.DEPIN_BP || 4000);
  const bpLiq = Number(process.env.LIQ_BP || 2000);
  const bpTsy = Number(process.env.TSY_BP || 2000);

  if (!depinTreasury || !liq || !tsy) throw new Error("one or more revenue wallets missing");
  if (bpRaf + bpDepin + bpLiq + bpTsy !== 10000) throw new Error("bps do not sum to 100
percent");

  const AXIOMRevenueRouter = await ethers.getContractFactory("AXIOMRevenueRouter");
  const rev = await AXIOMRevenueRouter.deploy(
    admin, axm, rafAddr, depinTreasury, liq, tsy,
    bpRaf, bpDepin, bpLiq, bpTsy
  );
  await rev.waitForDeployment();
  const revAddr = await rev.getAddress();
  console.log(`AXIOMRevenueRouter at ${revAddr}`);
  saveAddress(network.name, "AXIOMRevenueRouter", revAddr, {
    rafBp: bpRaf, depinBp: bpDepin, liqBp: bpLiq, tsyBp: bpTsy
  });
}
main().catch((e)=>{ console.error(e); process.exit(1); });
```

config/addresses.json template after first run

```
{
  "peaq": {
    "AXM": { "address": "0x...", "vault": "0x...", "updatedAt": "..." },
    "EmissionsVault": { "address": "0x...", "start": "0", "duration": "31536000", "beneficiary": "0x...",
"updatedAt": "..." },
    "DIDRegistry": { "address": "0x...", "updatedAt": "..." },
    "DePINNodeRegistry": { "address": "0x...", "schemaLicense": "0x...", "updatedAt": "..." },
    "DePINRewardRouter": { "address": "0x...", "schemaKyc": "0x...", "updatedAt": "..." },
    "RealEstateAcquisitionFund": { "address": "0x...", "updatedAt": "..." },
    "AXIOMRevenueRouter": { "address": "0x...", "rafBp": 2000, "depinBp": 4000, "liqBp": 2000,
"tsyBp": 2000, "updatedAt": "..." }
  }
}
```

Run sequence

1. npm i

2. copy env sample to .env and fill values

3. npm run compile

4. npm run deploy:axm

5. npm run deploy:vault

6. npm run deploy:did

7. npm run deploy:depin

8. npm run deploy:revenue

Post deploy actions

top up EmissionsVault by transferring AXM from your distribution vault and calling approve plus vault.topUp if you use transferFrom

grant DID issuer roles to your KYC and node license issuers

fund DePINRewardRouter with AXM for the first epoch

set node weights in DePINNodeRegistry if needed

route initial protocol AXM into AXIOMRevenueRouter and call route to test splits