# **Activity 3 — Build, Deploy & Operate a Production-Style ERC-20 on DIDLab**

## **What you’ll do (today)**

1. Scaffold a Hardhat **v3 (ESM)** project with **Viem**.
2. Write a gas-aware ERC-20 (**cap, pause, roles, batch airdrop**).
3. Deploy to **your team’s DIDLab chain**.
4. Interact by scripts (transfer, approve, **airdrop**, logs & gas stats).
5. Use **MetaMask** on your DIDLab network.

✅ **Requirements (student laptop):** Node **22.x** (LTS), npm, Git, VS Code, MetaMask.

❌ Don’t install hardhat-gas-reporter today (it conflicts with Hardhat v3).

## **0) Map your team → DIDLab network**

Pick your team number (01–12) and set these **once per shell**:

# >>> Replace 01 with your real team number <<<  
TEAM=01  
  
# RPC URL & CHAIN ID for your team  
export RPC\_URL="https://hh-${TEAM}.didlab.org"  
  
# Chain IDs are sequential from 31337  
# 01→31337, 02→31338, ... 12→31348  
export CHAIN\_ID=$((31336 + 10#$TEAM))  
  
# Quick sanity: should return 200 OK and 0x7a69 for team 01  
curl -s -H 'content-type: application/json' \  
 -d '{"jsonrpc":"2.0","method":"eth\_chainId","params":[],"id":1}' \  
 "$RPC\_URL" | jq -r .result

You’ll also use **one private key** (from your team’s 20 accounts) as the deployer:

# WARNING: class faucet key only. Don't use a real wallet key.  
export PRIVATE\_KEY="0x<your\_team\_private\_key\_no\_quotes>"

## **1) Project scaffold**

mkdir -p ~/didlab-activity3 && cd ~/didlab-activity3  
  
# Node must be 22.x  
node -v  
  
npm init -y  
  
# Hardhat v3 + Viem template deps (ESM)  
npm i -D hardhat@^3 \  
 @nomicfoundation/hardhat-toolbox-viem@^5 \  
 @nomicfoundation/hardhat-ignition@^3 \  
 typescript@~5.8.0 \  
 viem@^2.30.0 \  
 [@types/node@^22.8.5](mailto:@types/node@^22.8.5)# App deps  
npm i dotenv @openzeppelin/contracts@^5

Initialize Hardhat:

npx hardhat --init  
# Choose:  
# Version: hardhat-3  
# Project type: node-test-runner-viem  
# ESM? -> Yes

Create a .env (you’ll paste your values here):

cat > .env <<'EOF'  
RPC\_URL=  
CHAIN\_ID=  
PRIVATE\_KEY=  
# Optional deploy args (can be changed later)  
TOKEN\_NAME=CampusCredit  
TOKEN\_SYMBOL=CAMP  
TOKEN\_CAP=2000000 # tokens (human), 18 decimals  
TOKEN\_INITIAL=1000000 # tokens to deployer (human)  
EOF

Populate env with your shell values:

# Append your current shell vars into .env  
{  
 echo "RPC\_URL=$RPC\_URL"  
 echo "CHAIN\_ID=$CHAIN\_ID"  
 echo "PRIVATE\_KEY=$PRIVATE\_KEY"  
} >> .env

## **2) Hardhat config (ESM, Viem, DIDLab network)**

hardhat.config.ts

import "dotenv/config";  
import { defineConfig } from "hardhat/config";  
import "@nomicfoundation/hardhat-toolbox-viem";  
  
const RPC\_URL = process.env.RPC\_URL || "";  
const CHAIN\_ID = process.env.CHAIN\_ID ? Number(process.env.CHAIN\_ID) : undefined;  
  
export default defineConfig({  
 solidity: {  
 version: "0.8.24",  
 settings: { optimizer: { enabled: true, runs: 200 } },  
 },  
 networks: {  
 didlab: {  
 type: "http",  
 url: RPC\_URL,  
 chainId: CHAIN\_ID,  
 },  
 },  
});

## **3) Write the contract**

contracts/CampusCreditV2.sol

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.24;  
  
import {ERC20} from "[@openzeppelin/contracts/token/ERC20/ERC20.sol](mailto:@openzeppelin/contracts/token/ERC20/ERC20.sol)";  
import {ERC20Burnable} from "[@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol](mailto:@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol)";  
import {ERC20Capped} from "[@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol](mailto:@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol)";  
import {ERC20Pausable} from "[@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol](mailto:@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol)";  
import {AccessControl} from "[@openzeppelin/contracts/access/AccessControl.sol](mailto:@openzeppelin/contracts/access/AccessControl.sol)";  
  
/\*\*  
 \* CampusCreditV2  
 \* - Cap enforced on mint  
 \* - Pausable transfers  
 \* - Roles: ADMIN, MINTER, PAUSER  
 \* - Batch airdrop (gas-aware), custom errors  
 \*/  
contract CampusCreditV2 is ERC20, ERC20Burnable, ERC20Capped, ERC20Pausable, AccessControl {  
 bytes32 public constant MINTER\_ROLE = keccak256("MINTER\_ROLE");  
 bytes32 public constant PAUSER\_ROLE = keccak256("PAUSER\_ROLE");  
  
 error CapExceeded();  
 error ArrayLengthMismatch();  
  
 constructor(  
 string memory name\_,  
 string memory symbol\_,  
 uint256 cap\_, // in wei (18 decimals)  
 address initialReceiver,  
 uint256 initialMint // in wei  
 )  
 ERC20(name\_, symbol\_)  
 ERC20Capped(cap\_)  
 {  
 \_grantRole(DEFAULT\_ADMIN\_ROLE, msg.sender);  
 \_grantRole(MINTER\_ROLE, msg.sender);  
 \_grantRole(PAUSER\_ROLE, msg.sender);  
  
 if (initialMint > 0) {  
 \_mint(initialReceiver, initialMint);  
 }  
 }  
  
 function pause() external onlyRole(PAUSER\_ROLE) { \_pause(); }  
 function unpause() external onlyRole(PAUSER\_ROLE) { \_unpause(); }  
  
 function mint(address to, uint256 amount) external onlyRole(MINTER\_ROLE) {  
 \_mint(to, amount);  
 }  
  
 function airdrop(address[] calldata to, uint256[] calldata amounts) external onlyRole(MINTER\_ROLE) {  
 if (to.length != amounts.length) revert ArrayLengthMismatch();  
  
 uint256 len = to.length;  
 uint256 sum;  
 for (uint256 i = 0; i < len; ) {  
 sum += amounts[i];  
 unchecked { ++i; }  
 }  
 if (totalSupply() + sum > cap()) revert CapExceeded();  
  
 for (uint256 j = 0; j < len; ) {  
 \_mint(to[j], amounts[j]);  
 unchecked { ++j; }  
 }  
 }  
  
 // OZ v5 combines hooks via \_update  
 function \_update(address from, address to, uint256 value)  
 internal  
 override(ERC20, ERC20Pausable, ERC20Capped)  
 {  
 super.\_update(from, to, value);  
 }  
}

Compile:

npx hardhat compile

## **4) Deploy script (Viem, EIP-1559)**

scripts/deploy.ts

import "dotenv/config";  
import { artifacts } from "hardhat";  
import { createWalletClient, createPublicClient, http, parseUnits, getAddress  
} from "viem";  
import { privateKeyToAccount } from "viem/accounts";  
  
const RPC\_URL = process.env.RPC\_URL!;  
const CHAIN\_ID = Number(process.env.CHAIN\_ID!);  
const PRIVATE\_KEY = (process.env.PRIVATE\_KEY || "").replace(/^0x/, "");  
  
const NAME = process.env.TOKEN\_NAME || "CampusCredit";  
const SYMBOL = process.env.TOKEN\_SYMBOL || "CAMP";  
const CAP\_HUMAN = process.env.TOKEN\_CAP || "2000000";  
const INIT\_HUMAN = process.env.TOKEN\_INITIAL || "1000000";  
  
async function main() {  
 if (!RPC\_URL || !CHAIN\_ID || !PRIVATE\_KEY) throw new Error("Missing env vars");  
  
 const { abi, bytecode } = await artifacts.readArtifact("CampusCreditV2");  
  
 const chain = {  
 id: CHAIN\_ID,  
 name: `didlab-${CHAIN\_ID}`,  
 nativeCurrency: { name: "ETH", symbol: "ETH", decimals: 18 },  
 rpcUrls: { default: { http: [RPC\_URL] } },  
 } as const;  
  
 const account = privateKeyToAccount(`0x${PRIVATE\_KEY}`);  
 const wallet = createWalletClient({ account, chain, transport: http(RPC\_URL) });  
 const publicClient = createPublicClient({ chain, transport: http(RPC\_URL) });  
  
 const cap = parseUnits(CAP\_HUMAN, 18n);  
 const initialMint = parseUnits(INIT\_HUMAN, 18n);  
  
 console.log("Deploying CampusCreditV2…");  
 const hash = await wallet.deployContract({  
 abi, bytecode,  
 args: [NAME, SYMBOL, cap, getAddress(account.address), initialMint],  
 maxPriorityFeePerGas: 2\_000\_000\_000n, // 2 gwei  
 maxFeePerGas: 20\_000\_000\_000n, // 20 gwei  
 });  
 console.log("Deploy tx:", hash);  
  
 const rcpt = await publicClient.waitForTransactionReceipt({ hash });  
 console.log("Deployed at:", rcpt.contractAddress);  
 console.log("Block:", rcpt.blockNumber);  
  
 // Save for later scripts (optional)  
 console.log(`\nAdd this to .env:\nTOKEN\_ADDRESS=${rcpt.contractAddress}\n`);  
}  
  
main().catch((e) => { console.error(e); process.exit(1); });

Deploy:

npx hardhat run scripts/deploy.ts --network didlab  
# Copy the printed TOKEN\_ADDRESS into your .env:  
# echo "TOKEN\_ADDRESS=0x..." >> .env

## **5) Interaction scripts**

### **A) Transfer & Approve**

scripts/transfer-approve.ts

import "dotenv/config";  
import { artifacts } from "hardhat";  
import { createWalletClient, createPublicClient, http,  
 parseUnits, formatUnits, getAddress  
} from "viem";  
import { privateKeyToAccount } from "viem/accounts";  
  
const RPC\_URL = process.env.RPC\_URL!;  
const CHAIN\_ID = Number(process.env.CHAIN\_ID!);  
const PRIVATE\_KEY = (process.env.PRIVATE\_KEY || "").replace(/^0x/, "");  
const TOKEN = process.env.TOKEN\_ADDRESS as `0x${string}`;  
  
const RECIPIENT = process.env.RECIPIENT || ""; // optional teammate address  
  
async function main() {  
 if (!RPC\_URL || !CHAIN\_ID || !PRIVATE\_KEY || !TOKEN) throw new Error("Missing env");  
  
 const { abi } = await artifacts.readArtifact("CampusCreditV2");  
 const chain = { id: CHAIN\_ID, name: `didlab-${CHAIN\_ID}`, nativeCurrency: { name:"ETH",symbol:"ETH",decimals:18 }, rpcUrls:{ default:{ http:[RPC\_URL] } } } as const;  
  
 const account = privateKeyToAccount(`0x${PRIVATE\_KEY}`);  
 const wallet = createWalletClient({ account, chain, transport: http(RPC\_URL) });  
 const publicClient = createPublicClient({ chain, transport: http(RPC\_URL) });  
  
 const me = getAddress(account.address);  
 const you = RECIPIENT ? getAddress(RECIPIENT) : me; // fallback self  
  
 const bal = async (label:string) => {  
 const bMe = await publicClient.readContract({ address: TOKEN, abi, functionName:"balanceOf", args:[me] }) as bigint;  
 const bYou= await publicClient.readContract({ address: TOKEN, abi, functionName:"balanceOf", args:[you] }) as bigint;  
 console.log(`${label} | Me: ${formatUnits(bMe,18)} CAMP | You: ${formatUnits(bYou,18)} CAMP`);  
 };  
  
 await bal("Before");  
  
 // Transfer 100 CAMP (lower tip)  
 const tx1 = await wallet.writeContract({  
 address: TOKEN, abi, functionName: "transfer", args: [you, parseUnits("100", 18n)],  
 maxPriorityFeePerGas: 1\_000\_000\_000n, maxFeePerGas: 20\_000\_000\_000n  
 });  
 const r1 = await publicClient.waitForTransactionReceipt({ hash: tx1 });  
 console.log("transfer tx:", tx1, "gasUsed:", r1.gasUsed.toString());  
  
 // Approve 50 CAMP  
 const tx2 = await wallet.writeContract({  
 address: TOKEN, abi, functionName: "approve", args: [you, parseUnits("50", 18n)],  
 maxPriorityFeePerGas: 2\_000\_000\_000n, maxFeePerGas: 21\_000\_000\_000n  
 });  
 const r2 = await publicClient.waitForTransactionReceipt({ hash: tx2 });  
 console.log("approve tx:", tx2, "gasUsed:", r2.gasUsed.toString());  
  
 // Show allowance  
 const alw = await publicClient.readContract({  
 address: TOKEN, abi, functionName: "allowance", args: [me, you]  
 }) as bigint;  
 console.log("allowance:", formatUnits(alw, 18), "CAMP");  
  
 await bal("After");  
}  
  
main().catch((e) => { console.error(e); process.exit(1); });

Run:

# Optionally set a teammate address for RECIPIENT  
# echo "RECIPIENT=0x<their\_address>" >> .env  
  
npx hardhat run scripts/transfer-approve.ts --network didlab

### **B) Batch Airdrop + Gas Compare**

scripts/airdrop.ts

import "dotenv/config";  
import { artifacts } from "hardhat";  
import { createWalletClient, createPublicClient, http, parseUnits, getAddress  
} from "viem";  
import { privateKeyToAccount } from "viem/accounts";  
  
const RPC\_URL = process.env.RPC\_URL!;  
const CHAIN\_ID = Number(process.env.CHAIN\_ID!);  
const PRIVATE\_KEY = (process.env.PRIVATE\_KEY || "").replace(/^0x/, "");  
const TOKEN = process.env.TOKEN\_ADDRESS as `0x${string}`;  
  
async function main() {  
 if (!RPC\_URL || !CHAIN\_ID || !PRIVATE\_KEY || !TOKEN) throw new Error("Missing env");  
  
 const { abi } = await artifacts.readArtifact("CampusCreditV2");  
 const chain = { id: CHAIN\_ID, name:`didlab-${CHAIN\_ID}`, nativeCurrency:{ name:"ETH",symbol:"ETH",decimals:18 }, rpcUrls:{ default:{ http:[RPC\_URL] } } } as const;  
  
 const account = privateKeyToAccount(`0x${PRIVATE\_KEY}`);  
 const wallet = createWalletClient({ account, chain, transport: http(RPC\_URL) });  
 const publicClient = createPublicClient({ chain, transport: http(RPC\_URL) });  
  
 // >>> Paste 3–6 recipient addresses for your team here (teammates + self ok)  
 const recipients = [  
 getAddress(account.address),  
 // "0x.................",  
 // "0x.................",  
 ];  
 const amounts = recipients.map(() => parseUnits("10", 18n));  
  
 // One batch airdrop  
 const txBatch = await wallet.writeContract({  
 address: TOKEN, abi, functionName: "airdrop", args: [recipients, amounts],  
 maxPriorityFeePerGas: 2\_000\_000\_000n, maxFeePerGas: 22\_000\_000\_000n  
 });  
 const rBatch = await publicClient.waitForTransactionReceipt({ hash: txBatch });  
 const feeBatch = rBatch.gasUsed \* (rBatch.effectiveGasPrice ?? 0n);  
 console.log("Airdrop:", txBatch, "gasUsed:", rBatch.gasUsed.toString(), "fee(wei):", feeBatch.toString());  
  
 // N individual transfers (compare)  
 let totalGas = 0n, totalFee = 0n;  
 for (let i=0; i<recipients.length; i++) {  
 const tx = await wallet.writeContract({  
 address: TOKEN, abi, functionName: "transfer", args: [recipients[i], amounts[i]],  
 maxPriorityFeePerGas: 2\_000\_000\_000n, maxFeePerGas: 22\_000\_000\_000n  
 });  
 const r = await publicClient.waitForTransactionReceipt({ hash: tx });  
 totalGas += r.gasUsed;  
 totalFee += r.gasUsed \* (r.effectiveGasPrice ?? 0n);  
 }  
 console.log("Singles total gasUsed:", totalGas.toString(), "fee(wei):", totalFee.toString());  
  
 if (totalGas > 0n) {  
 const saved = (Number(totalGas - rBatch.gasUsed) / Number(totalGas) \* 100).toFixed(2);  
 console.log(`Batch saved ≈ ${saved}% gas vs singles`);  
 }  
}  
  
main().catch((e) => { console.error(e); process.exit(1); });

Run:

npx hardhat run scripts/airdrop.ts --network didlab

### **C) Logs & Events (Transfer / Approval)**

scripts/logs-query.ts

import "dotenv/config";  
import { artifacts } from "hardhat";  
import { createPublicClient, http, decodeEventLog } from "viem";  
  
const RPC\_URL = process.env.RPC\_URL!;  
const CHAIN\_ID = Number(process.env.CHAIN\_ID!);  
const TOKEN = process.env.TOKEN\_ADDRESS as `0x${string}`;  
  
async function main() {  
 if (!RPC\_URL || !CHAIN\_ID || !TOKEN) throw new Error("Missing env");  
  
 const { abi } = await artifacts.readArtifact("CampusCreditV2");  
 const chain = { id: CHAIN\_ID, name:`didlab-${CHAIN\_ID}`, nativeCurrency:{name:"ETH",symbol:"ETH",decimals:18}, rpcUrls:{default:{http:[RPC\_URL]}} } as const;  
 const publicClient = createPublicClient({ chain, transport: http(RPC\_URL) });  
  
 const latest = await publicClient.getBlockNumber();  
 const fromBlock = latest > 2000n ? latest - 2000n : 0n;  
  
 const logs = await publicClient.getLogs({ address: TOKEN, fromBlock, toBlock: "latest" });  
  
 for (const log of logs) {  
 try {  
 const ev = decodeEventLog({ abi, data: log.data, topics: log.topics });  
 console.log(`[${log.blockNumber}] ${ev.eventName}`, ev.args);  
 } catch {  
 /\* non-erc20 events (if any) \*/  
 }  
 }  
}  
  
main().catch((e) => { console.error(e); process.exit(1); });

Run:

npx hardhat run scripts/logs-query.ts --network didlab

## **6) MetaMask (custom network + token)**

1. **Add Network** → *Add a network manually*
   1. Network Name: DIDLab Team XX
   2. RPC URL: <https://hh-XX.didlab.org>
   3. Chain ID (decimal): your CHAIN\_ID
   4. Currency Symbol: ETH
2. **Import Account** (class faucet private key).
3. **Import Token** → paste your TOKEN\_ADDRESS.
4. Send a small transfer in MetaMask; capture the tx hash.

Run logs-query.ts and confirm the Transfer event shows up.

## **7) NPM scripts (optional)**

Add to package.json:

{  
 "scripts": {  
 "compile": "hardhat compile",  
 "deploy": "hardhat run scripts/deploy.ts --network didlab",  
 "xfer": "hardhat run scripts/transfer-approve.ts --network didlab",  
 "airdrop": "hardhat run scripts/airdrop.ts --network didlab",  
 "logs": "hardhat run scripts/logs-query.ts --network didlab"  
 }  
}

Then you can run:

npm run deploy  
npm run xfer  
npm run airdrop  
npm run logs

## **8) Quick Troubleshooting**

* **Node 18 warning / crashes** → Install & use Node **22.x**.
* **HHE15: Invalid discriminator / network type** → type: "http" must be set under networks.didlab.
* **HHE909 / solc mismatch** → Keep solidity.version = "0.8.24"; re-compile.
* **BigInt errors** → Use parseUnits("100", 18n) (note the 18n).
* **“returned no data (0x)”** → You’re calling the token at a **non-contract** address or wrong TOKEN\_ADDRESS. Re-deploy or fix .env.
* **RPC mismatch** → Verify curl $RPC\_URL with eth\_chainId.

## **9) What to hand in (for assignment later)**

* TOKEN\_ADDRESS, deploy block, your roles & cap.
* A **screenshot** of MetaMask with your custom network + token.
* **Console output** from airdrop.ts (batch vs singles gas).
* A short note on **why** your airdrop is gas-aware (custom errors, calldata, unchecked loops, single tx amortization).