Convention-Over-Configuration Implementation Plan

Overview

Implement a convention-over-configuration approach to dramatically reduce setup complexity and improve developer experience. This approach will provide sensible defaults while maintaining full customization capabilities.

Current State vs. Target State

Current State (Complex Configuration)

```
// audityzer.config.js - Current
module.exports = {
  chains: {
    ethereum: {
      rpcUrl: process.env.ETHEREUM_RPC_URL,
      chainId: 1,
      gasLimit: 8000000,
      gasPrice: 'auto',
      accounts: [process.env.PRIVATE_KEY]
    },
    polygon: {
      rpcUrl: process.env.POLYGON_RPC_URL,
      chainId: 137,
      gasLimit: 6000000,
      gasPrice: 'auto',
      accounts: [process.env.PRIVATE_KEY]
    }
  },
  testing: {
    vulnerabilities: ['reentrancy', 'oracle', 'flashloan'],
    wallet: 'metamask',
    timeout: 30000,
    retries: 3
  },
  reporting: {
    format: 'json',
    outputDir: './reports',
    includeTimestamp: true
  }
};
```

Target State (Convention-Based)

```
// aduityzer.config.js - Simplified
module.exports = {
  // Only specify what differs from conventions
  chain: 'ethereum', // Default: ethereum
  tests: ['reentrancy', 'oracle'], // Default: all available
  // Everything else uses smart defaults
};
```

Implementation Strategy

1. Smart Defaults System

Chain Configuration Conventions

```
// Default chain configurations
const CHAIN_DEFAULTS = {
  ethereum: {
    rpcUrl: () => process.env.ETHEREUM_RPC_URL || 'https://eth.llamarpc.com',
    chainId: 1,
    gasLimit: 8000000,
    blockTime: 12000
  },
  polygon: {
    rpcUrl: () => process.env.POLYGON_RPC_URL || 'https://polygon.llamarpc.com',
    chainId: 137,
    gasLimit: 6000000,
    blockTime: 2000
  },
  arbitrum: {
    rpcUrl: () => process.env.ARBITRUM_RPC_URL || 'https://arb1.arbitrum.io/rpc',
    chainId: 42161,
    gasLimit: 10000000,
    blockTime: 1000
  }
};
```

Project Structure Conventions

```
my-audit-project/
├── aduityzer.config.js     # Optional - only for overrides
├── .env                    # Auto-detected environment variables
├── tests/                  # Auto-discovered test directory
│   ├── security/           # Security tests (auto-run)
│   ├── integration/        # Integration tests
│   └── unit/               # Unit tests
├── contracts/              # Auto-discovered contract directory
├── reports/                # Auto-generated reports directory
└── package.json            # Project metadata
```

Test Discovery Conventions

```
// Auto-discovery patterns
const TEST_PATTERNS = {
  security: [
    'tests/**/*.security.js',
    'tests/security/**/*.test.js',
    'security/**/*.test.js'
  ],
  integration: [
    'tests/**/*.integration.js',
    'tests/integration/**/*.test.js'
  ],
  aa: [
    'tests/**/*.aa.js',
    'tests/aa/**/*.test.js',
    'aa-tests/**/*.test.js'
  ]
};
```

2. Environment-Based Configuration

Automatic Environment Detection

```
// Auto-detect environment and apply appropriate defaults
const detectEnvironment = () => {
  if (process.env.NODE_ENV === 'production') return 'mainnet';
  if (process.env.CI) return 'ci';
  if (process.env.HARDHAT_NETWORK) return 'local';
  return 'development';
};

const ENVIRONMENT_DEFAULTS = {
  development: {
    mockMode: true,
    verbose: true,
    timeout: 60000
  },
  ci: {
    mockMode: false,
    verbose: false,
    timeout: 30000,
    format: 'json'
  },
  production: {
    mockMode: false,
    verbose: false,
    timeout: 120000,
    format: 'html'
  }
};
```

Smart Environment Variable Detection

```
// Auto-detect common environment variable patterns
const detectEnvVars = () => {
  const config = {};

  // RPC URLs
  Object.keys(process.env).forEach(key => {
    if (key.endsWith('_RPC_URL')) {
      const chain = key.replace('_RPC_URL', '').toLowerCase();
      config.chains = config.chains || {};
      config.chains[chain] = { rpcUrl: process.env[key] };
    }
  });

  // API Keys
  if (process.env.PIMLICO_API_KEY) {
    config.aa = { pimlicoApiKey: process.env.PIMLICO_API_KEY };
  }

  return config;
};
```

3. Intelligent Configuration Merging

Configuration Hierarchy

```
const buildConfig = () => {
  const configs = [
    getGlobalDefaults(),           // 1. Global defaults
    getEnvironmentDefaults(),      // 2. Environment-specific defaults
    getProjectConventions(),       // 3. Project structure conventions
    getEnvVarConfig(),            // 4. Environment variables
    getUserConfig(),              // 5. User configuration file
    getCliOverrides()             // 6. CLI argument overrides
  ];

  return deepMerge(...configs);
};
```

Smart Merging Logic

```
const deepMerge = (...configs) => {
  return configs.reduce((merged, config) => {
    Object.keys(config).forEach(key => {
      if (Array.isArray(config[key])) {
        // Arrays: merge unique values
        merged[key] = [...new Set([...(merged[key] || []), ...config[key]])];
      } else if (typeof config[key] === 'object' && config[key] !== null) {
        // Objects: recursive merge
        merged[key] = deepMerge(merged[key] || {}, config[key]);
      } else {
        // Primitives: override
        merged[key] = config[key];
      }
    });
    return merged;
  }, {});
};
```

File Structure Conventions

Automatic Project Detection

```
const detectProjectType = () => {
  const packageJson = readPackageJson();
  const files = listProjectFiles();

  // Detect based on dependencies
  if (packageJson.dependencies?.['@account-abstraction/contracts']) {
    return 'aa-wallet';
  }
  if (packageJson.dependencies?.['@openzeppelin/contracts']) {
    return 'defi-protocol';
  }
  if (files.includes('hardhat.config.js')) {
    return 'hardhat-project';
  }
  if (files.includes('foundry.toml')) {
    return 'foundry-project';
  }

  return 'general';
};
```

Template-Based Defaults

```
const PROJECT_TEMPLATES = {
  'aa-wallet': {
    tests: ['aa-userop', 'aa-paymaster', 'aa-bundler'],
    chains: ['ethereum', 'polygon'],
    features: ['account-abstraction']
  },
  'defi-protocol': {
    tests: ['reentrancy', 'oracle', 'flashloan', 'access-control'],
    chains: ['ethereum'],
    features: ['defi-testing']
  },
  'bridge-protocol': {
    tests: ['bridge-security', 'cross-chain'],
    chains: ['ethereum', 'polygon', 'arbitrum'],
    features: ['cross-chain-testing']
  }
};
```

Configuration API Design

Minimal Configuration Interface

```
// audityzer.config.js - Minimal
module.exports = {
  // Only specify what you want to change
  chain: 'polygon',           // Default: 'ethereum'
  tests: ['reentrancy'],      // Default: all available
  wallet: 'coinbase',         // Default: 'metamask'
  format: 'html'              // Default: 'json'
};
```

Progressive Configuration

```
// audityzer.config.js - Progressive complexity
module.exports = {
  // Level 1: Basic overrides
  chain: 'ethereum',

  // Level 2: Feature-specific config
  aa: {
    bundler: 'pimlico',
    addon: 'social-recovery'
  },

  // Level 3: Advanced customization
  chains: {
    ethereum: {
      rpcUrl: 'https://my-custom-rpc.com',
      gasLimit: 10000000
    }
  },

  // Level 4: Full control
  testing: {
    vulnerabilities: {
      reentrancy: {
        enabled: true,
        severity: 'high',
        customRules: ['./custom-reentrancy-rules.js']
      }
    }
  }
};
```

Configuration Validation

```
const validateConfig = (config) => {
  const errors = [];
  const warnings = [];

  // Validate chain configuration
  if (config.chain && !SUPPORTED_CHAINS.includes(config.chain)) {
    errors.push(`Unsupported chain: ${config.chain}`);
  }

  // Validate test types
  if (config.tests) {
    const invalidTests = config.tests.filter(test => !AVAILABLE_TESTS.includes(test));
    if (invalidTests.length > 0) {
      warnings.push(`Unknown test types: ${invalidTests.join(', ')}`);
    }
  }

  // Validate AA configuration
  if (config.aa?.bundler && !SUPPORTED_BUNDLERS.includes(config.aa.bundler)) {
    errors.push(`Unsupported bundler: ${config.aa.bundler}`);
  }

  return { errors, warnings };
};
```

Migration Strategy

Automatic Migration Tool

```
// scripts/migrate-config.js
const migrateConfig = async () => {
  const oldConfig = await readOldConfig();
  const newConfig = transformToConventions(oldConfig);

  // Show what will change
  console.log('Migration Preview:');
  console.log('Old config:', JSON.stringify(oldConfig, null, 2));
  console.log('New config:', JSON.stringify(newConfig, null, 2));

  const confirm = await askConfirmation('Apply migration?');
  if (confirm) {
    await writeNewConfig(newConfig);
    await backupOldConfig(oldConfig);
    console.log('  Migration completed!');
  }
};
```

Backward Compatibility

```
const loadConfig = () => {
  // Try new convention-based config first
  if (fs.existsSync('adityzer.config.js')) {
    const userConfig = require('./adityzer.config.js');
    return applyConventions(userConfig);
  }

  // Fall back to legacy config format
  if (fs.existsSync('adityzer.legacy.config.js')) {
    console.warn('⚠ Using legacy config format. Run `adityzer migrate` to update.');
    return loadLegacyConfig();
  }

  // Use pure conventions
  return applyConventions({});
};
```

Implementation Timeline

Phase 1: Core Convention System (Week 1-2)

- [ ] Implement smart defaults system
- [ ] Create configuration hierarchy
- [ ] Build environment detection
- [ ] Add project type detection

Phase 2: File Structure Conventions (Week 3)

- [ ] Implement auto-discovery patterns
- [ ] Create template-based defaults
- [ ] Add intelligent merging logic

Phase 3: Migration Tools (Week 4)

- [ ] Build migration utility
- [ ] Implement backward compatibility
- [ ] Create validation system

Phase 4: Testing & Documentation (Week 5)

- [ ] Comprehensive testing
- [ ] Update documentation
- [ ] Create migration guides

Success Metrics

Developer Experience Improvements

- **Setup Time**: Reduce from 15 minutes to 2 minutes

- **Configuration Lines**: Reduce from 50+ lines to 5-10 lines
- **Error Rate**: Reduce configuration errors by 80%
- **Time to First Test**: Reduce from 30 minutes to 5 minutes

Adoption Metrics

- **New User Completion Rate**: Increase from 60% to 90%
- **Configuration Support Issues**: Reduce by 70%
- **Documentation Complexity**: Reduce by 50%

Technical Implementation

Core Convention Engine

```
// src/core/conventions/index.js
class ConventionEngine {
  constructor() {
    this.defaults = new DefaultsManager();
    this.detector = new ProjectDetector();
    this.merger = new ConfigMerger();
  }

  async buildConfig(userConfig = {}) {
    const projectType = await this.detector.detectProjectType();
    const environment = this.detector.detectEnvironment();
    const envVars = this.detector.detectEnvVars();

    const baseConfig = this.defaults.getDefaults(projectType, environment);

    return this.merger.merge([
      baseConfig,
      envVars,
      userConfig
    ]);
  }
}
```

Configuration Schema

```
// src/core/conventions/schema.js
const CONFIG_SCHEMA = {
  chain: {
    type: 'string',
    enum: ['ethereum', 'polygon', 'arbitrum', 'optimism', 'base'],
    default: 'ethereum'
  },
  tests: {
    type: 'array',
    items: {
      type: 'string',
      enum: ['reentrancy', 'oracle', 'flashloan', 'access-control', 'front-running']
    },
    default: () => getAllAvailableTests()
  },
  wallet: {
    type: 'string',
    enum: ['metamask', 'coinbase', 'walletconnect', 'mock'],
    default: 'metamask'
  },
  aa: {
    type: 'object',
    properties: {
      enabled: { type: 'boolean', default: false },
      bundler: { type: 'string', enum: ['pimlico', 'stackup', 'alchemy'], default: 'pim
lico' },
      addon: { type: 'string', enum: ['social-recovery', 'session-keys'], default:
null }
    }
  }
};
```

This convention-over-configuration system will dramatically improve the developer experience while maintaining the flexibility that advanced users need. The key is providing intelligent defaults that work for 80% of use cases while allowing full customization when needed.