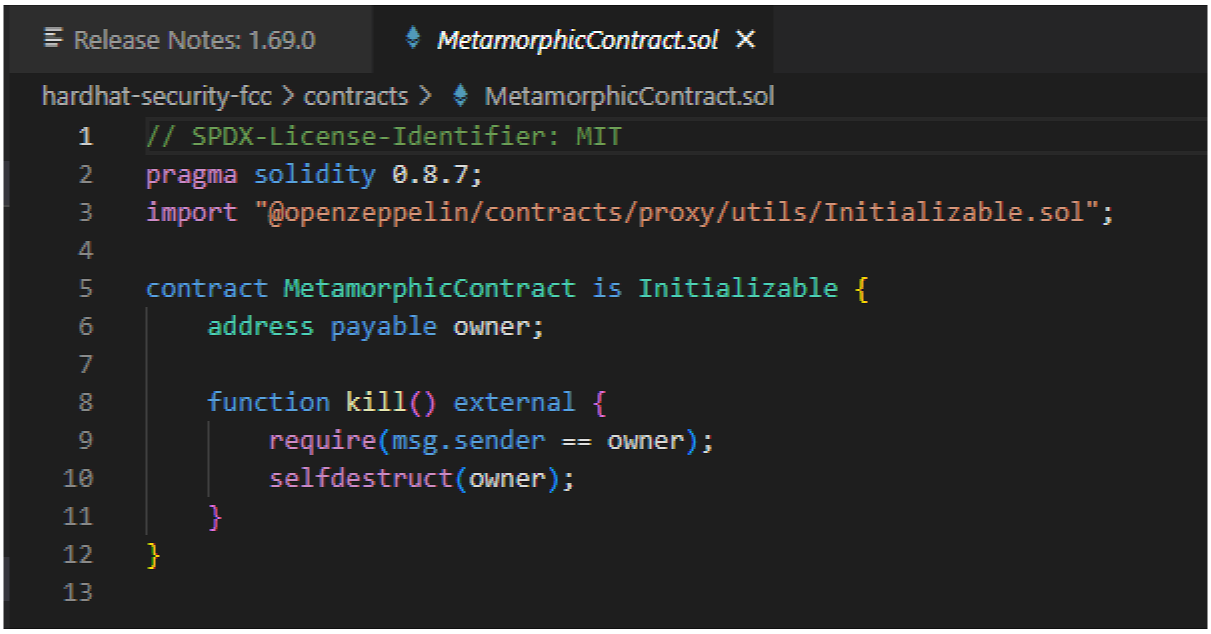
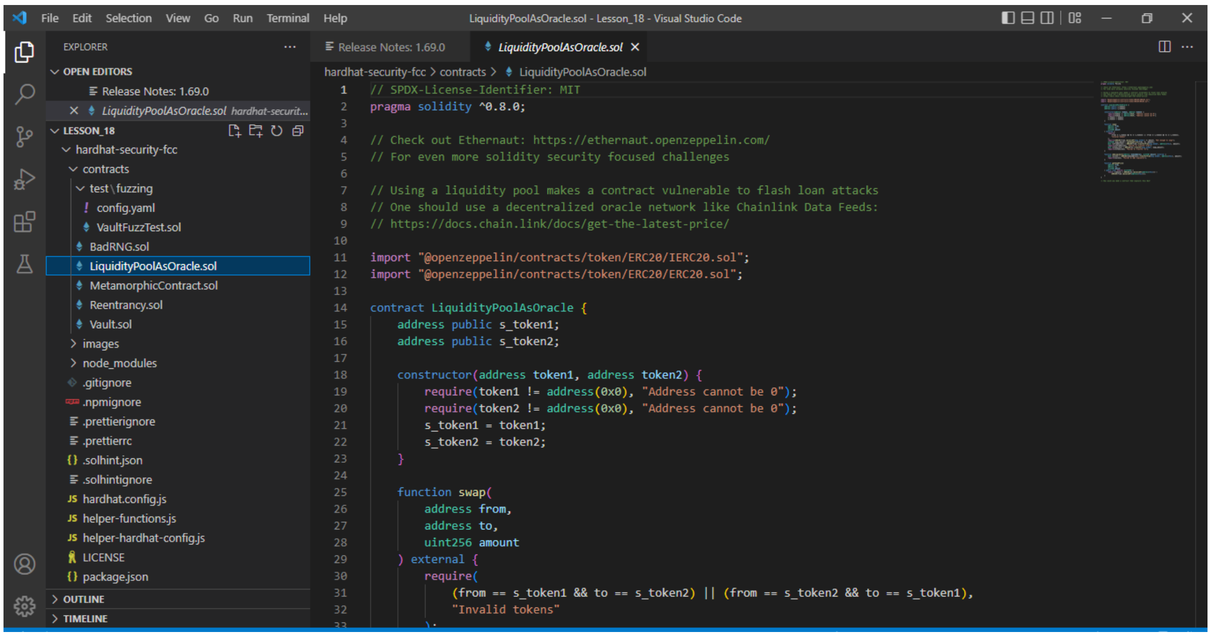
- MetamorphicContract



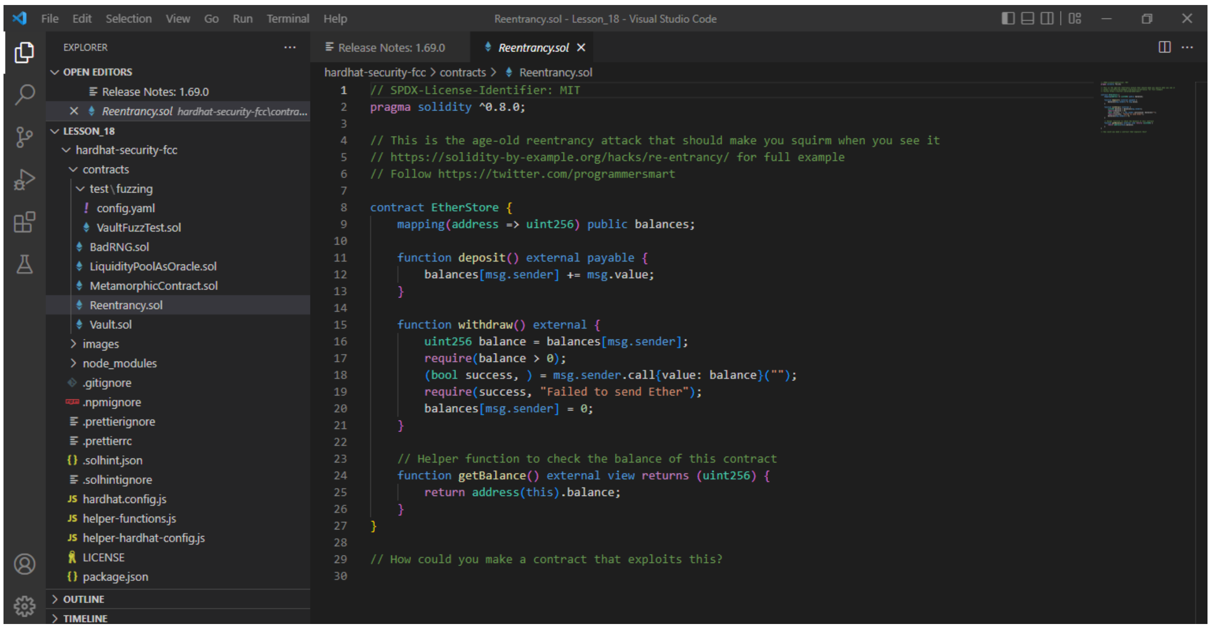
```solidity
// SPDX-License-Identifier: MIT
pragma solidity 0.8.7;
import "@openzeppelin/contracts/proxy/utils/Initializable.sol";

contract MetamorphicContract is Initializable {
    address payable owner;

    function kill() external {
        require(msg.sender == owner);
        selfdestruct(owner);
    }
}

```

- LiquidityPoolAsOracle



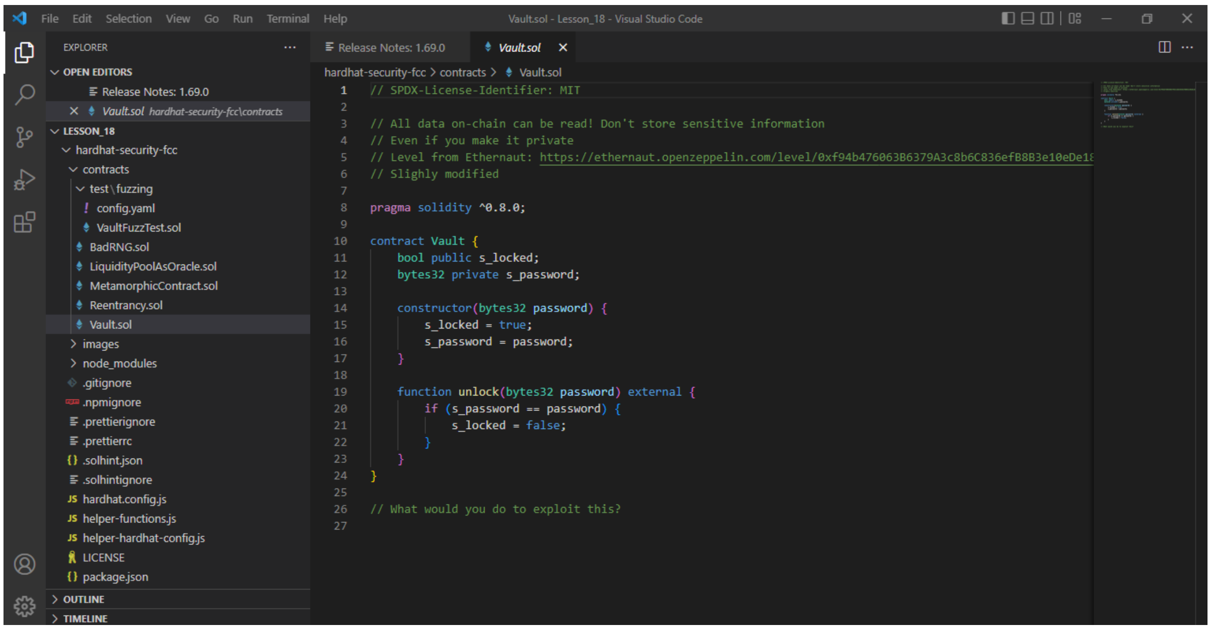
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// Check out Ethernaut: https://ethernaut.openzeppelin.com/
// For even more solidity security focused challenges

// Using a liquidity pool makes a contract vulnerable to flash loan attacks
// One should use a decentralized oracle network like Chainlink Data Feeds:
// https://docs.chain.link/docs/get-the-latest-price/

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

contract LiquidityPoolAsOracle {
    address public s_token1;
    address public s_token2;

    constructor(address token1, address token2) {
        require(token1 != address(0x0), "Address cannot be 0");
        require(token2 != address(0x0), "Address cannot be 0");
        s_token1 = token1;
        s_token2 = token2;
    }

    function swap(
        address from,
        address to,
        uint256 amount
    ) external {
        require(
            (from == s_token1 && to == s_token2) || (from == s_token2 && to == s_token1),
            "Invalid tokens"
```

- Reentrancy



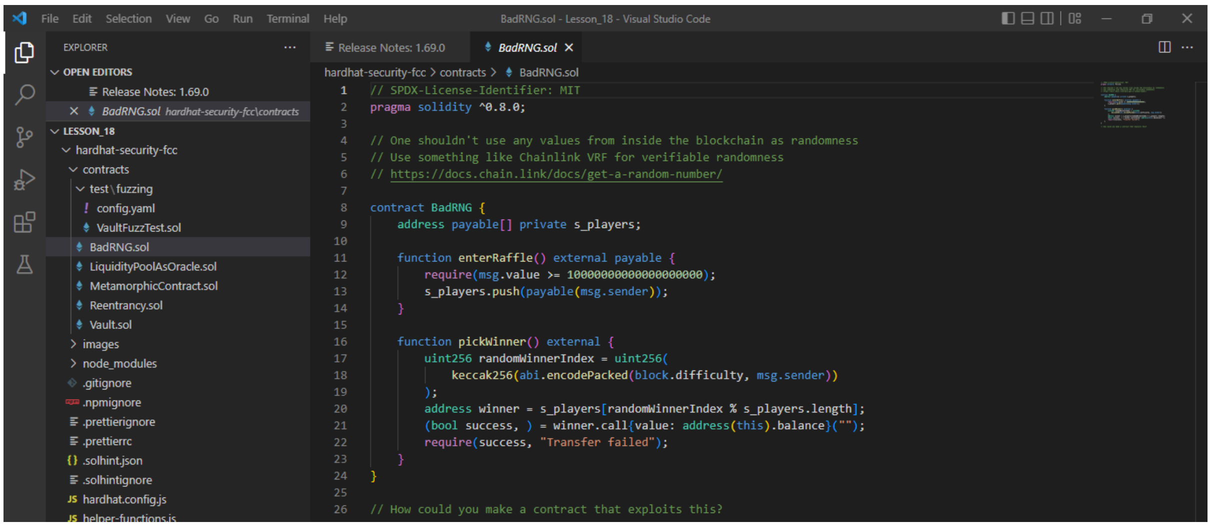
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// This is the age-old reentrancy attack that should make you squirm when you see it
// https://solidity-by-example.org/hacks/re-entrancy/ for full example
// Follow https://twitter.com/programmersmart

contract EtherStore {
    mapping(address => uint256) public balances;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 balance = balances[msg.sender];
        require(balance > 0);
        (bool success, ) = msg.sender.call{value: balance}("");
        require(success, "Failed to send Ether");
        balances[msg.sender] = 0;
    }

    // Helper function to check the balance of this contract
    function getBalance() external view returns (uint256) {
        return address(this).balance;
    }
}

// How could you make a contract that exploits this?
```

- Vault



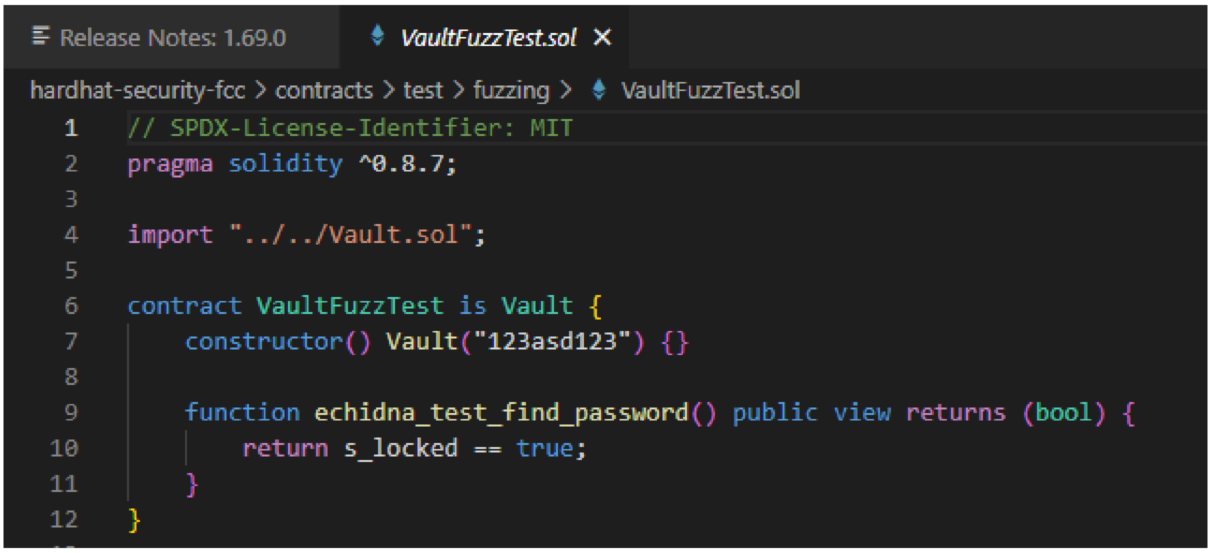
```solidity
// SPDX-License-Identifier: MIT

// All data on-chain can be read! Don't store sensitive information
// Even if you make it private
// Level from Ethernaut: https://ethernaut.openzeppelin.com/level/0xf94b476063B6379A3c8b6C836efB8B3e10eDe18
// Slighly modified

pragma solidity ^0.8.0;

contract Vault {
    bool public s_locked;
    bytes32 private s_password;

    constructor(bytes32 password) {
        s_locked = true;
        s_password = password;
    }

    function unlock(bytes32 password) external {
        if (s_password == password) {
            s_locked = false;
        }
    }
}

// What would you do to exploit this?
```

- BadRNG



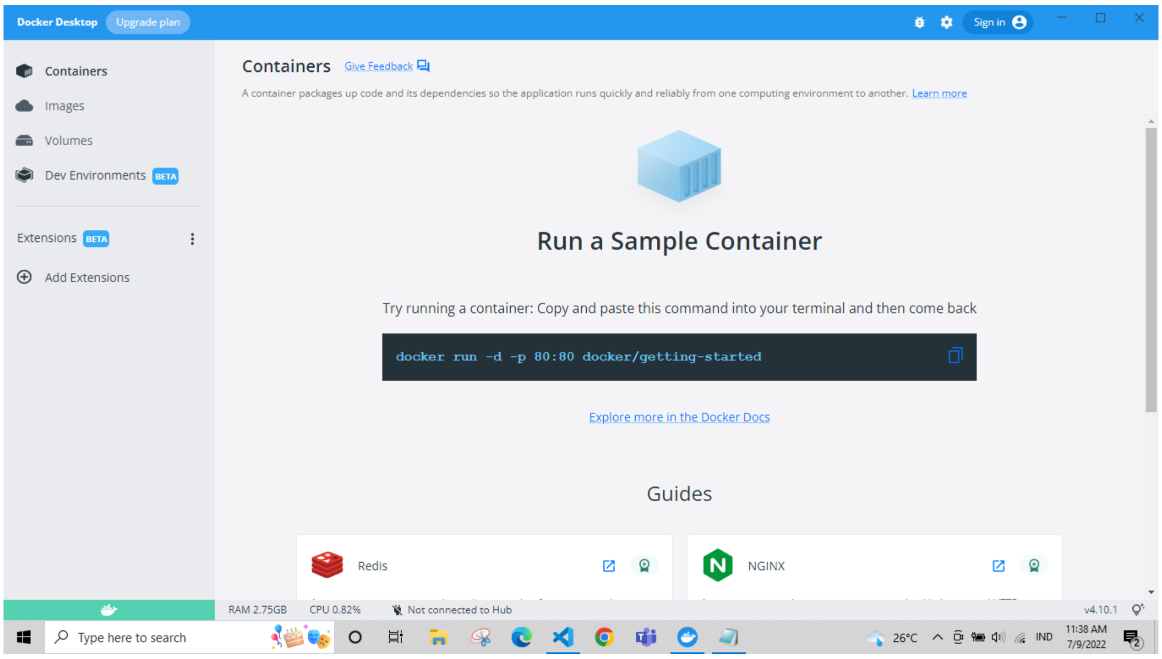
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// One shouldn't use any values from inside the blockchain as randomness
// Use something like Chainlink VRF for verifiable randomness
// https://docs.chain.link/docs/get-a-random-number/

contract BadRNG {
    address payable[] private s_players;

    function enterRaffle() external payable {
        require(msg.value >= 1000000000000000000);
        s_players.push(payable(msg.sender));
    }

    function pickWinner() external {
        uint256 randomWinnerIndex = uint256(
            keccak256(abi.encodePacked(block.difficulty, msg.sender))
        );
        address winner = s_players[randomWinnerIndex % s_players.length];
        (bool success, ) = winner.call{value: address(this).balance}("");
        require(success, "Transfer failed");
    }
}

// How could you make a contract that exploits this?
```

- VaultFuzzTest


Release Notes: 1.69.0
VaultFuzzTest.sol
hardhat-security-fcc > contracts > test > fuzzing > VaultFuzzTest.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "../../Vault.sol";

contract VaultFuzzTest is Vault {
    constructor() Vault("123asd123") {}

    function echidna_test_find_password() public view returns (bool) {
        return s_locked == true;
    }
}
```

- Docker



A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

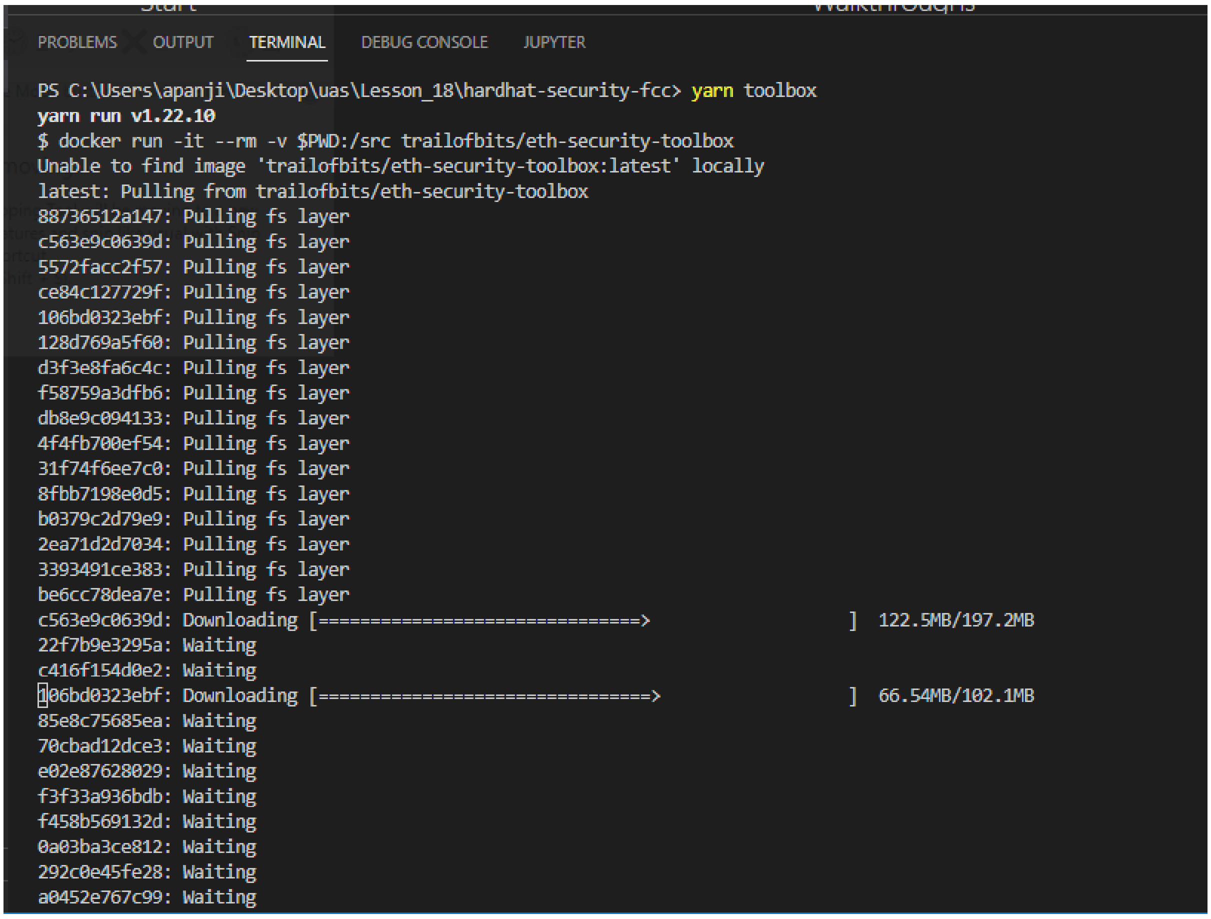
Run a Sample Container

Try running a container: Copy and paste this command into your terminal and then come back

```
docker run -d -p 80:80 docker/getting-started
```

Explore more in the Docker Docs

- Toolbox



```
PS C:\Users\apanji\Desktop\uas\Lesson_18\hardhat-security-fcc> yarn toolbox
yarn run v1.22.10
$ docker run -it --rm -v $PWD:/src trailofbits/eth-security-toolbox
Unable to find image 'trailofbits/eth-security-toolbox:latest' locally
latest: Pulling from trailofbits/eth-security-toolbox
88736512a147: Pulling fs layer
c563e9c0639d: Pulling fs layer
5572facc2f57: Pulling fs layer
ce84c127729f: Pulling fs layer
106bd0323ebf: Pulling fs layer
128d769a5f60: Pulling fs layer
d3f3e8fa6c4c: Pulling fs layer
f58759a3dfb6: Pulling fs layer
db8e9c094133: Pulling fs layer
4f4fb700ef54: Pulling fs layer
31f74f6ee7c0: Pulling fs layer
8fbb7198e0d5: Pulling fs layer
b0379c2d79e9: Pulling fs layer
2ea71d2d7034: Pulling fs layer
3393491ce383: Pulling fs layer
be6cc78dea7e: Pulling fs layer
c563e9c0639d: Downloading [===============================>                   ]  122.5MB/197.2MB
22f7b9e3295a: Waiting
c416f154d0e2: Waiting
106bd0323ebf: Downloading [================================>                  ]  66.54MB/102.1MB
85e8c75685ea: Waiting
70cbad12dce3: Waiting
e02e87628029: Waiting
f3f33a936bdb: Waiting
f458b569132d: Waiting
0a03ba3ce812: Waiting
292c0e45fe28: Waiting
a0452e767c99: Waiting
```