GTSL - Project Setup Documentation

This page provides an overview of the setup implemented for the development of the Global Trust Service Status List (GTSL) that is part of the Future Trust project. It also specifies the technologies and methodologies used to implement the setup.

Tools overview

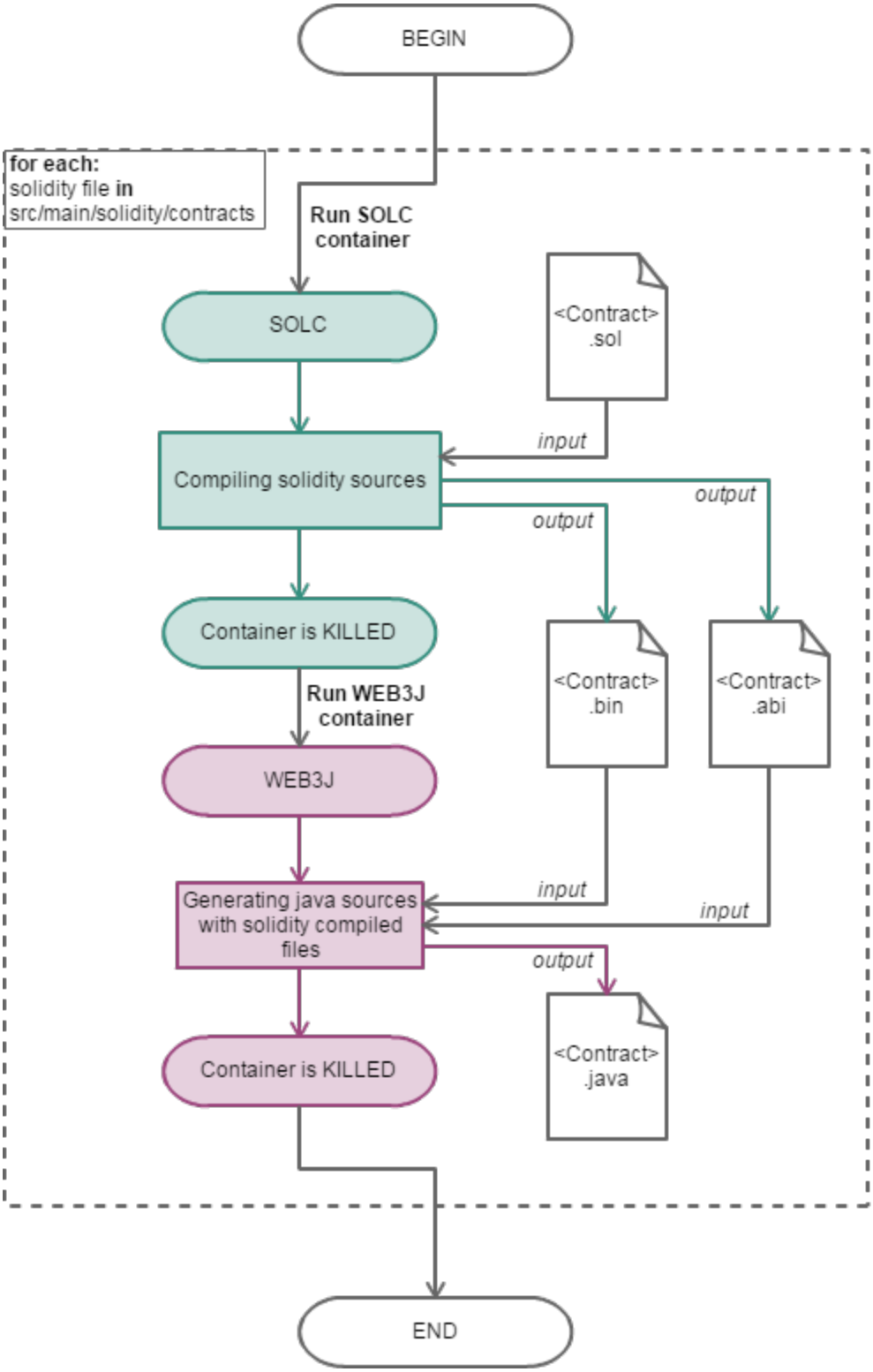
Tool	Version	Host	Description
Java	1.8	Local	programming language
Solidity	0.4.11	None (configured in solidity file itself)	smart-contracts language
Spring Boot	1.5.4.RELEASE	None (configured in pom.xml)	framework designed to simplify the development of Spring applications
Maven	3.5.0	Local	software project management and comprehension tool
Docker	17.05.0-ce	Local	open platform to build, ship, and run distributed applications
Docker-compose	1.13.0	Local	tool for defining and running multi-container Docker applications
Docker-machine	0.11.0	Local (required only on Windows & Mac OS)	tool that lets you install Docker Engine on virtual hosts
Ethereum	v1.6.5	Docker container	decentralized platform based on a blockchain that runs smart contracts
IPFS	v0.4.9	Docker container	protocol designed to create a permanent and decentralized file system
Solc	stable	Docker container	Solidity compiler
Web3j Command Line Tool	v2.2.1	Docker container	tool for generating Java sources from Solidity binaries

Project setup

The GTSL uses Ethereum as a ledger to keep track of the state of each trusted list it contains. On the other hand, the GTSL uses IPFS in order to store data regarding each trusted list. The application itself is a Spring Boot application.

Build smart-contracts

In the project, smart-contracts are written in Solidity and will be used to define the behaviour of the ledger. In order to use those smart-contracts in the application, it is needed that they are generated as Java classes. To achieve this, a process of compiling and generating has been implemented as described in the figure below. The implementation is a bash script that uses Docker containers for the Solc compiler and the Web3j generator.

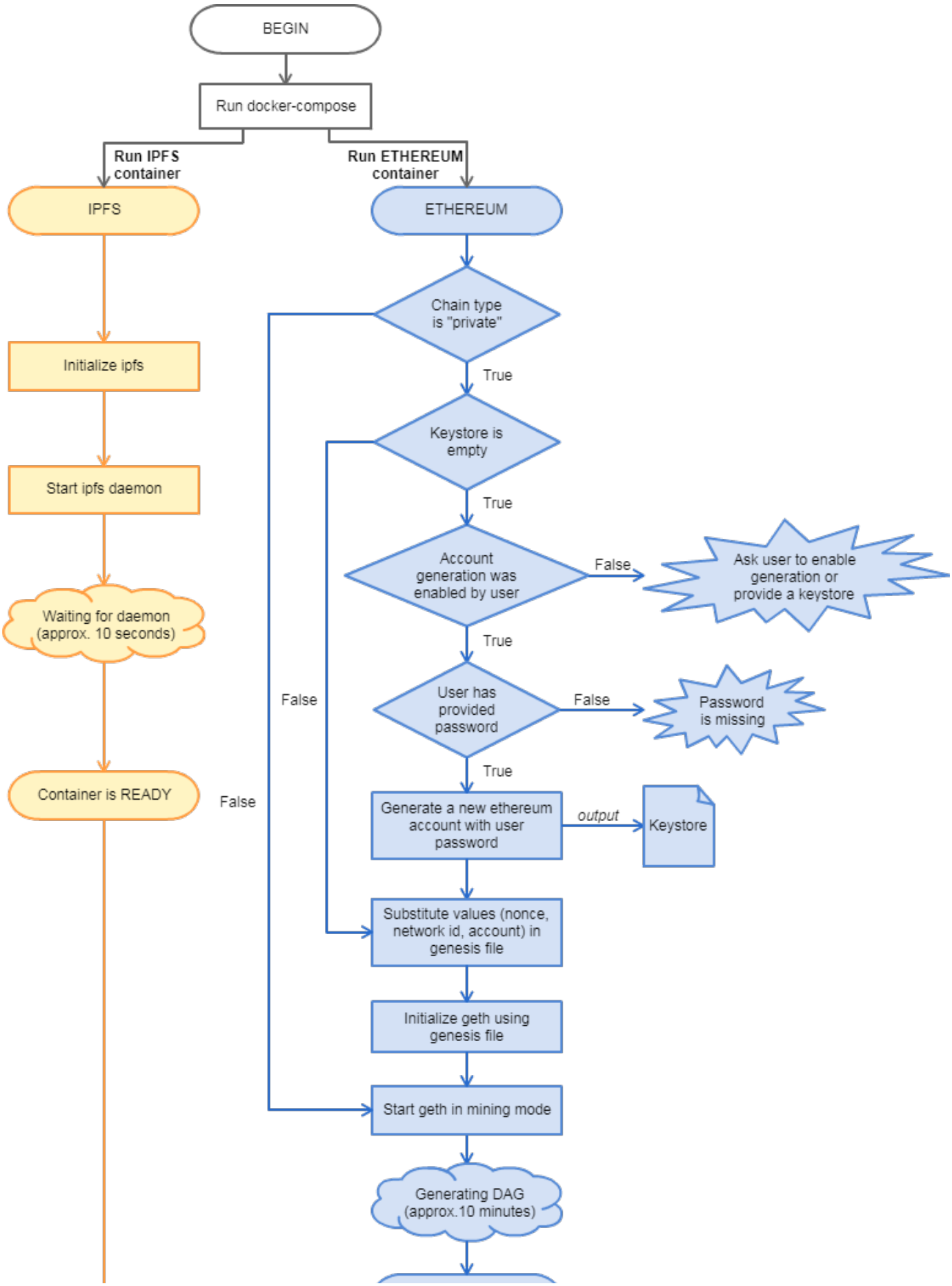


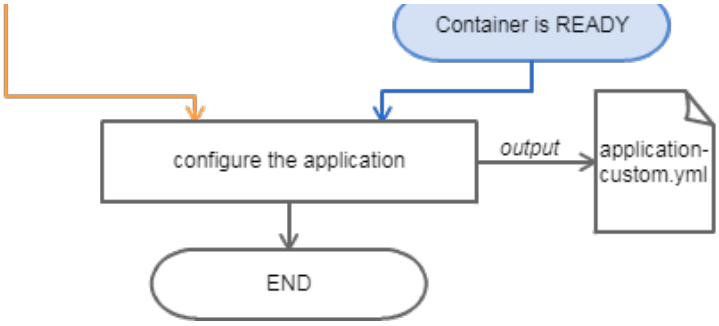
The process is divided into two parts and both parts are applied to all solidity files. First, the script runs a SOLC container that takes as input the solidity source file and creates a binary file and a .abi file which described the smart-contract. Then, those two files are used as input for the WEB3J container that generates a java source file for the smart-contract.

Warning: Only the smart-contracts in the folder *src/main/solidity/contracts* are generated. Those smart-contracts are "*final*" smart-contracts (final in the sense that there is no smart-contract which extends it). If you define *abstract* contracts, you have to put them in a sub-folder.

Run the dependencies

For the application to work it is needed that an IPFS node and an Ethereum node are up. The process described in the figure below explains how it is achieved using Docker containers. Before the application starts, the two containers have to be ready. It means an IPFS daemon is started and a private Ethereum blockchain is generated in mining mode. Then, some files are generated into the **outputs** folder. Those files have to be copied into the **gtsl** project.


BEGIN
Run docker-compose
Run IPFS container
Run ETHEREUM container
IPFS
ETHEREUM
Initialize ipfs
Start ipfs daemon
Waiting for daemon (approx. 10 seconds)
Container is READY
False
Chain type is "private"
True
Keystore is empty
True
Account generation was enabled by user
False
Ask user to enable generation or provide a keystore
True
User has provided password
False
Password is missing
False
True
Generate a new ethereum account with user password
output
Keystore
Substitute values (nonce, network id, account) in genesis file
Initialize geth using genesis file
Start geth in mining mode
Generating DAG (approx.10 minutes)


Container is READY
configure the application
output
application-custom.yml
END