Per l'installazione di Quorum:

1. Installa VirtualBox
2. Installa Vagrant
3. Creare una cartella di lavoro: mkdir work
4. Entrare nella cartella: cd work
5. Nel caso si utilizzi Linux, eseguire: git clone https://github.com/jpmorganchase/quorum-examples
6. Nel caso si utilizzi Windows, scaricare quorum-examples all'interno della cartella di lavoro, cliccando "clone or download" da https://github.com/jpmorganchase/quorum-examples
7. Entrare nella cartella appena creata quorum-examples
8. Eseguire: vagrant up
9. Eseguire: vagrant ssh
10. Al termine dell'esecuzione del comando ci si deve trovare in una console linux:

    vagrant@ubuntu-xenial:~$
11. In caso contrario eseguire nuovamente il punto (8) e poi (9)
12. All'interno della console linux lanciare: cd quorum-examples/7nodes
13. Eseguire: sudo ./raft-init.sh
14. Eseguire: sudo ./raft-start.sh
15. Per controllare che tutto sia stato fatto in modo corretto, eseguire: sudo geth attach qdata/dd1/geth.ipc
16. Nel caso in cui si apre una console per dd1 (nodo 1), tutto è corretto
17. Altrimenti lanciale: sudo ./stop.sh e ripartire dal punto (13)
18. Nel caso in cui continui a non aprirsi la console del nodo 1, chiudere il terminale, aprirne uno nuovo ed eseguire (3), (4), (7), (9) e da (12) in poi

Per i passi successivi, una volta che si è riusciti ad eseguire la console di un nodo:

1. Entrare nella console dei nodi 2 e 3: sudo geth attach qdata/dd2/geth.ipc e sudo geth attach qdata/dd3/geth.ipc
2. Eseguire in entrambi: eth.accounts[0]
3. Copiare i valori restituiti
4. Prendere il file del contratto Appalto.sol (si trova all'interno della cartella progetto/contracts) , nel costruttore, nella riga dove si trova direttore_lavori = INDIRIZZO_ACCOUNT, sostituire INDIRIZZO_ACCOUNT con il valore restituito nella console del nodo 2 (IMPORTANTE!!! togliere da questo valore i doppi apici, sia davanti che dietro, come si può vedere da INDIRIZZO_ACCOUNT presente nel file del contratto)
5. Fare la medesima cosa per l'indirizzo del nodo 3, solamente sostituirlo all'indirizzo di ditta

6. Lasciare aperto il terminale con la console aperta di un nodo, ad esempio del nodo 1: sudo geth attach qdata/dd1/geth.ipc

Per l'installazione dell'ambiente Truffle:

1. Aprire un nuovo terminale
2. Entrare nella cartella di lavoro: cd work
3. Installare Node.js e Npm (windows dal sito di Node.js; linux: sudo apt-get install nodejs; sudo apt-get install npm; npm init; npm install -g truffle)
4. Creare una cartella all'interno della cartella di lavoro per il progetto Truffle: mkdir progetto
5. Entrare nella cartella: cd progetto
6. Eseguire: truffle init (windows: truffle.cmd init)
7. All'interno della cartella "Tesi e tirocinio Blockchain-Iacopo Pacifici" ho già predisposto la cartella "progetto" con tutti i dettagli modificati. Occorre quindi sostituire la cartella progetto creata ed inizializzata in seguito alla truffle init con la cartella progetto già modificata da me. L'unica cosa da modificare sono gli indirizzi degli account per la ditta e direttore_lavori nel file Appalto.sol, come visto sopra. Fatto questo è possibile eseguire solamente il punto (12) e poi ripartire da (16). In caso si faccia manualmente, seguire i passi successivi
8. Modificare il file truffle.js con il codice seguente:

```
// File: `truffle.js`

module.exports = {
 networks: {
  development: {
   host: "127.0.0.1",
   port: 22000, //porta di un nodo lanciato nella blockchain
   network_id: "*", // Match any network id
   gasPrice: 0,
   gas: 4500000
  },
  nodetwo: {
   host: "127.0.0.1",
   port: 22001,
   network_id: "*", // Match any network id
   gasPrice: 0,
   gas: 4500000
```

```
  },
  nodethree: {
   host: "127.0.0.1",
   port: 22002,
   network_id: "*", // Match any network id
   gasPrice: 0,
   gas: 4500000
  }
 }
};
```

9. Entra nella cartella contracts: cd contracts
10. Crea nuovi file.sol per inserire i contratti (Appalto.sol, Conforme.sol, StringUtils.sol, Valore.sol) (Prendere questi file dalla cartella “Tesi e tirocinio Blockchain – Iacopo Pacifici/Contratti”)
11. Torna nella cartella del progetto: cd ..
12. Eseguire: truffle compile (windows: truffle.cmd compile)
13. Entra nella cartella migrations: cd migrations
14. Crea un file.js di migrazione per i contratti. È importante che contenga un numero inziale che sia diverso dal numero iniziale di eventuali altri file di migrazione già presenti nella cartella. Ad esempio: 3_deploy_contratto.js. Il file deve essere:

```
//// File: `./migrations/2_deploy_contratti.js`

var Appalto = artifacts.require("Appalto");
var Conforme = artifacts.require("Conforme");
var Valore = artifacts.require("Valore");
var StringUtils = artifacts.require("StringUtils");

module.exports = function(deployer) {

deployer.deploy(StringUtils).then(() => {

deployer.link(StringUtils,Appalto);
 return deployer.deploy(Appalto)
});
```

```
deployer.deploy(Conforme);

deployer.deploy(Valore);

};
```

15. Tornare nella cartella di progetto: cd ..
16. Eseguire: truffle migrate (windows: truffle.cmd migrate) e si avrà un output del tipo:

```
Running migration: 1_initial_migration.js
  Replacing Migrations...
  ... 0xf4dacd5d9ac2609bbb9fda41ae5afd3ffac5f678a9c55a18207ab6ce2f5d70f0
  Migrations: 0xe02da54df915c49fb7d6827062abd0a1b207c4b3
Saving successful migration to network...
  ... 0xd2c7c6a9d0e2ac1578d67c377c36614b60e1f906c03d2824c90d44edea06b94e
Saving artifacts...
Running migration: 2_deploy_contratti.js
  Deploying StringUtils...
  ... 0x46fd04718688dc27ccad40d3f2ec7e10c87684ddd69ab8dc1dbadcf14c18d5b3
  StringUtils: 0x0e857f78ea0c0e9b3111c74b881d0da778424d92
  Linking StringUtils to Appalto
  Deploying Appalto...
  ... 0x8a9b8ddb2135be3a76259156cc92ceaa7c7ff22cefbebce6309b54c6116f85ee
  Appalto: 0x8f9098360e8144c68ad87feceab72126115a78fe
  Deploying Conforme...
  ... 0x517a3a9dc806265c7dc76069e808e3dc4d71f1fb04bcc18cc6ccf5e98eca5269
  Conforme: 0x36ffd017f5bf85ca35d62b502fd7f7514bd172a0
  Deploying Valore...
  ... 0x6d36d1f47f0e8b89fc27feacc3f11f08e195427b4728c4e1c52372b9b4a5a2f4
```

Valore: 0x39ef4171f9677834ef97eddd9a60e648fa109f9f

Saving successful migration to network...

... 0xf78524a56693b3e9945813bb17b40cd27e3cfc05e9ec4c6c3ece727170e5750a

Saving artifacts...

17. Per verificare che il contratto sia stato deployato realmente, copiare un hash di transazione da questo output, ad esempio l'hash della transazione che ha creato il contratto Appalto:

    ... 0x8a9b8ddb2135be3a76259156cc92ceaa7c7ff22cefbebce6309b54c6116f85ee (ovviamente copiare solamente l'indirizzo senza i tre puntini iniziali)

18. Andare nel terminale lasciato aperto con la console del primo nodo e lanciare: eth.getTransactionReceipt("HASH_TRANSAZIONE_COPIATO_AL_PUNTO_17")

19. L'output dovrebbe restituire vari campi tra cui il campo contractAddress, ovvero l'indirizzo in cui è stato salvato il contratto Appalto

20. Come ulteriore controprova si può osservare che l'indirizzo di contractAddress corrisponde all'indirizzo restituito nell'output sopra:

    Appalto: 0x8f9098360e8144c68ad87feceab72126115a78fe

21. Dall'output sopra, copiare i valori dell'indirizzo in cui sono stati deployati i contratti. Ad esempio, Appalto: 0x8f9098360e8144c68ad87feceab72126115a78fe ← è l'indirizzo del contratto Appalto (fare per ogni contratto)

22. Entrare in build/contracts e per ogni contratto (escluso il contratto di default SimpleStorage se presente) copiare il valore di abi (sono oggetti json)

23. Utilizzare https://codebeautify.org/jsonminifier per minimizzare l'abi di ogni contratto

Testare che funzioni il contratto nella blockchain:

1. Tornare nel terminale lasciato aperto con la console del primo nodo

2. Eseguire: var abiAppalto = VALORE_ABI_DEL_CONTRATTO_APPALTO (è il valore che è stato creato attraverso il json minifier) (fare lo stesso per ogni contratto, cambiando il nome della variabile, ad esempio abiConforme, abiValore…)

3. Eseguire: var inidirizzoAppalto = "INDIRIZZO_DEL_CONTRATTO_APPALTO" (è il valore che avevamo ottenuto prima, dopo il lancio del truffle migrate. Importante, devono essere presenti i doppi apici, sia in testa che in coda) (fare lo stesso per ogni contratto, cambiando il nome della variabile, ad esempio indirizzoConforme, indirizzoValore…)

4. Eseguire: var istanzaAppalto = eth.contract(abiAppalto).at(indirizzoAppalto) (fare lo stesso per ogni contratto, cambiando le variabili a seconda del contratto di cui si va a creare l'istanza)

5. Ripetere lo stesso procedimento in altri 2 terminali con aperte altre due console di due nodi:

   sudo geth attach qdata/dd2/geth.ipc sudo geth attach qdata/dd3/geth.ipc

6. Per eseguire le funzioni del contratto è necessario eseguire nelle 3 console: eth.defaultAccount = eth.accounts[0]. Serve per impostare l'account di default che devono utilizzare i contratti

7. Fare la stessa cosa per gli altri contratti: Conforme.sol e Valore.sol
8. Ora è possibile eseguire le funzioni

NB: Nel caso in cui venga chiusa la console di un nodo è necessario riaprire la console e ripetere i passi nell'ordine: (2), (3), (4), (6), (7)

Esecuzione delle funzioni:

1. Settare prima gli indirizzi degli altri contratti. Ad esempio, per il contratto Appalto.sol devo settare gli indirizzi di Conforme.sol e Valore.sol. Per farlo utilizzare le funzioni setIndirizzoConforme e setIndirizzoValore. Eseguire:
   istanzaAppalto.setIndirizzoConforme("INDIRIZZO_CONTRATTO_CONFORME", {gas: 0x99999})

   Per le funzioni di setting è necessario specificare un gas per eseguire le operazioni

   INDIRIZZO_CONTRATTO_CONFORME viene preso dall'output ottenuto in seguito alla migrazione in Truffle
2. Fare la stessa cosa per settare l'indirizzo di Valore.sol, con la funzione setIndirizzoValore
3. Fare la stessa cosa nei contratti Valore.sol e Conforme.sol, utilizzando le funzioni setIndirizzoAppalto, setIndirizzoConforme, setIndirizzoValore.

   Ovviamente non è necessario utilizzare la funzione per settare l'indirizzo di un contratto nel contratto stesso. Ad esempio nel contratto Appalto.sol non devo settare il suo indirizzo attraverso la funzione setIndirizzoAppalto.
4. Inserire lavori e soglie. Eseguire: istanzaAppalto.addLavoro("NOME_LAVORO", {gas: 0x99999}). Eseguire: istanzaAppalto.addSoglia(COSTO_SOGLIA, PERCENTUALE_DA_RAGGIUNGERE, {gas: 0x99999}).
5. Per vedere se ha inserito il nuovo lavoro o la soglia eseguire rispettivamente: istanzaAppalto.numero_lavori() istanzaAppalto.numero_soglie()

   Nel caso in cui il numero dei lavori o delle soglie non è incrementato rispetto a prima, è possibile che sia stata eseguita la funzione da un account non corrispondente all'account del direttore dei lavori, quindi rieseguirla dall'account del direttore dei lavori (avendo seguito questa guida, l'account del direttore dei lavori coincide con l 'account del nodo 2, quindi eseguire le operazioni di addLavoro e addSoglia dal terminale con aperta la console del nodo 2)
6. A questo punto è possibile eseguire il resto delle funzioni

Funzioni eseguibili di Appalto.sol e come eseguirle:

- addLavoro("NOME_LAVORO_DA_INSERIRE", {gas: 0x99999}) ----Aggiunge un lavoro----
- addSoglia(COSTO_DA_PAGARE_AL_RAGGIUNGIMETO_SOGLIA, PERCENTUALE_DA_RAGGIUNGERE, {gas: 0x99999}) -----Aggiunge una soglia-----
- setIndirizzoConforme ("INDIRIZZO_CONTRATTO_CONFORME.SOL") ----Setta indirizzo del contratto Conforme.sol----
- setIndirizzoValore("INDIRIZZO_CONTRATTO_VALORE.SOL") ----Setta indirizzo del contratto Valore.sol----
- address() ------Restituisce l'indirizzo del contratto-------
- checkRimanente() -----Controlla se è rimasto del valore per terminare il contratto----
- checkValore() ------Controlla se il valore di completamento è inferiore o maggiore uguale della soglia stabilita------
- committenza() -----Restituisce l'indirizzo della committenza---
- direttore_lavori() -----Restituisce l'indirizzo del direttore dei lavori---
- ditta() -----Restituisce l'indirizzo della ditta---
- sta() ----Restituisce lo stato corrente-----
- totale() ----Restituisce il totale----
- valore() ---Restituisce il valore----
- numero_lavori() ----Restituisce il numero dei lavori----
- numero_soglie() ----Restituisce il numero delle soglie----
- soglia_corrente() ----Restituisce la soglia corrente----
- numero_requisiti() ----Restituisce il numero dei requisiti----
- lavori("NUMERO_LAVORO_GIA_INSERITO") ----Restituisce la tupla formata da [nome_lavoro, percentuale_completamento, percentuale_controllata]
- getBilancio() -----Restituisce il bilancio nel contratto----
- updateLavoro("NUMERO_LAVORO_DA_MODIFICARE", "PERCENTUALE_COMPLETAMENTO_ULTERIORE_RISPETTO_ALLA_PRECEDENTE", {gas: 0x99999}) ----Incrementa la percentuale di completamento del lavoro-----
- setRequisitoConformità("NUMERO_LAVORO", "REQUISITO_ZERO", "REQUISITO_UNO", "REQUISITO_DUE", …., "REQUISITO_OTTO") ----Setta i requisiti di conformità del lavoro a true o false---- E' possibile passare 1 o 0, oppure true o false.
- sendPagamento () ---Permette di pagare la ditta----
- ritiraFondi() --Permette alla committenza di ritirare i fondi inseriti nel contratto---
- killContratto() ----Termina il contratto---

Funzioni eseguibili di Valore.sol e come eseguirle:

- calcolaValore() ----Fa l'update del nuovo valore dei lavori rispetto allo stato precedente----
- setIndirizzoAppalto("INDIRIZZO_CONTRATTO_APPALTO") ----Setta l'indirizzo del contratto Appalto----
- setIndirizzoConforme("INDIRIZZO_CONTRATTO_CONFORME") ----Setta l'indirizzo del contratto Conforme----

Funzioni eseguibili di Conforme.sol e come eseguirle:

- setIndirizzoAppalto("INDIRIZZO_CONTRATTO_APPALTO") ----Setta l'indirizzo del contratto Appalto----
- setIndirizzoValore("INDIRIZZO_CONTRATTO_VALORE") ----Setta l'indirizzo del contratto Valore—
- checkConformitaLavoro() ---Controlla che tutti i requisiti di ogni lavoro siano conformi----