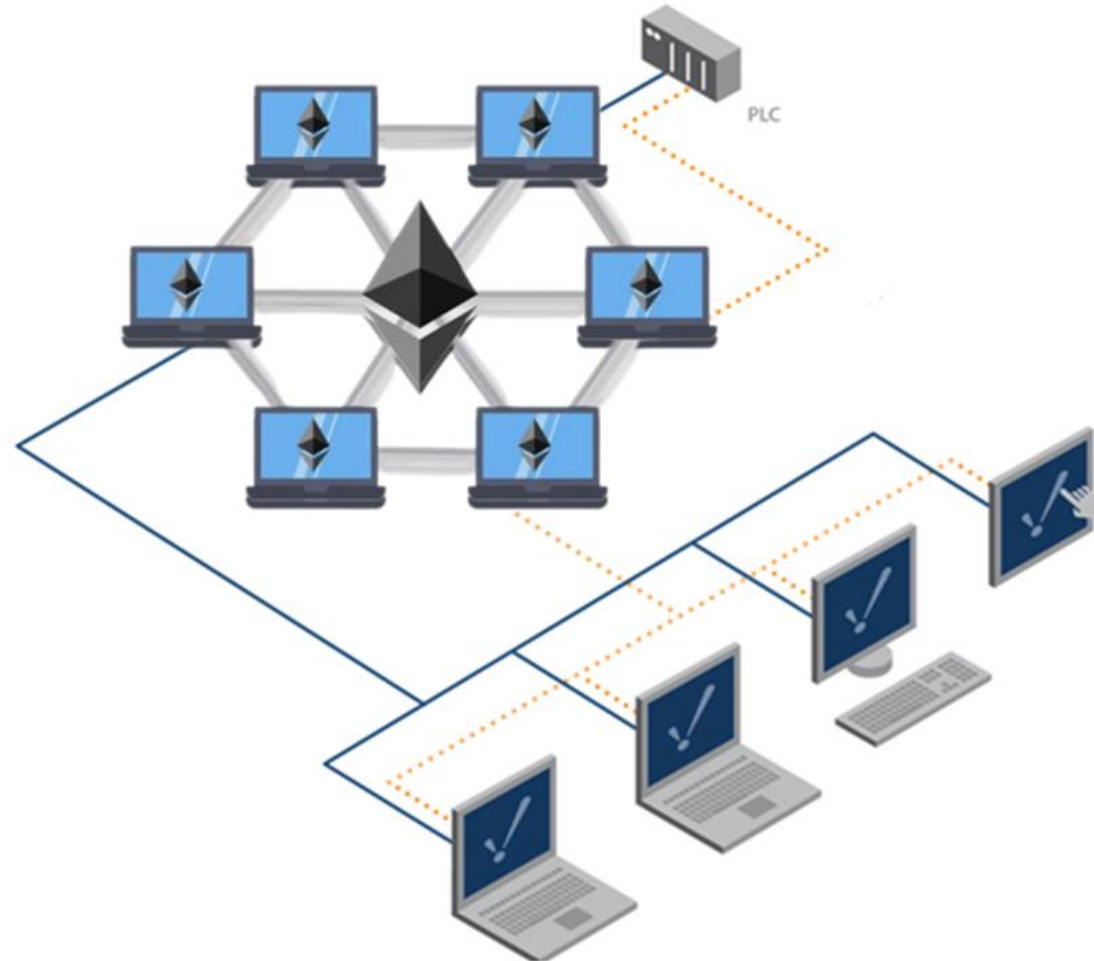


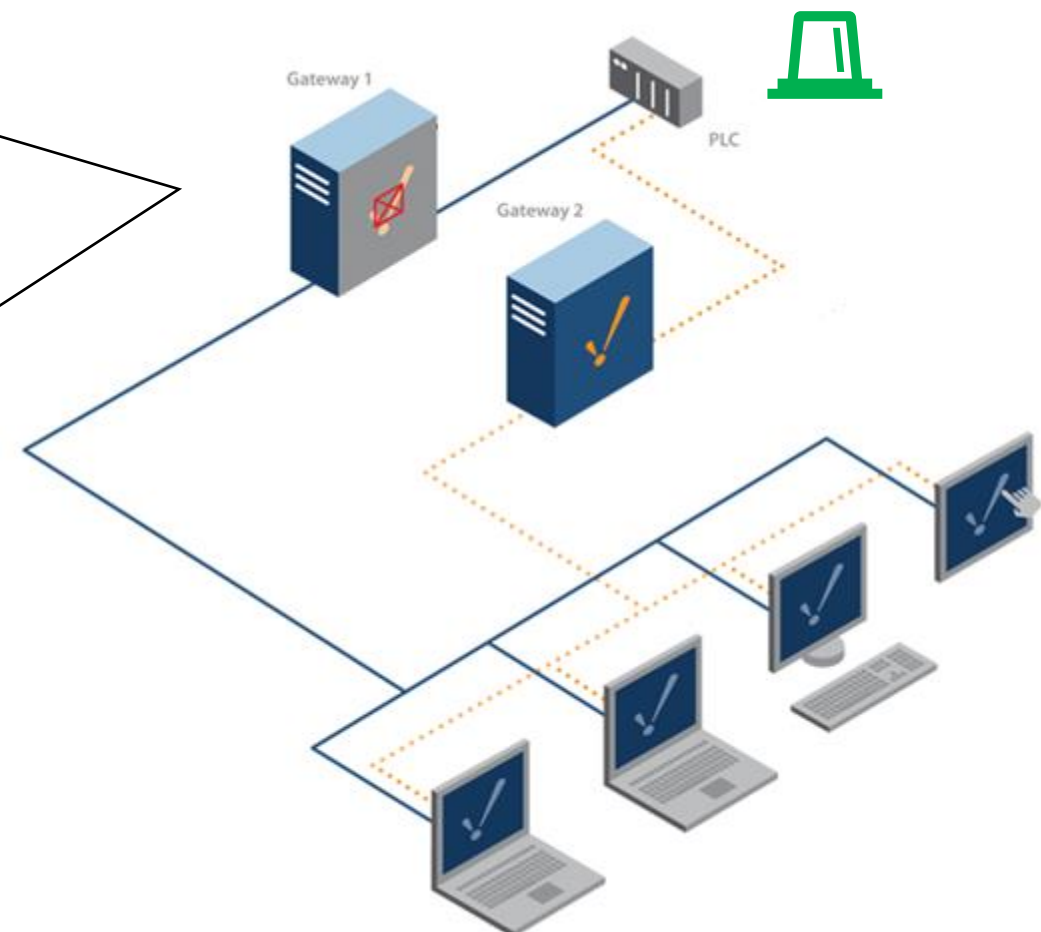
SCADA Descentralizado

V0.4



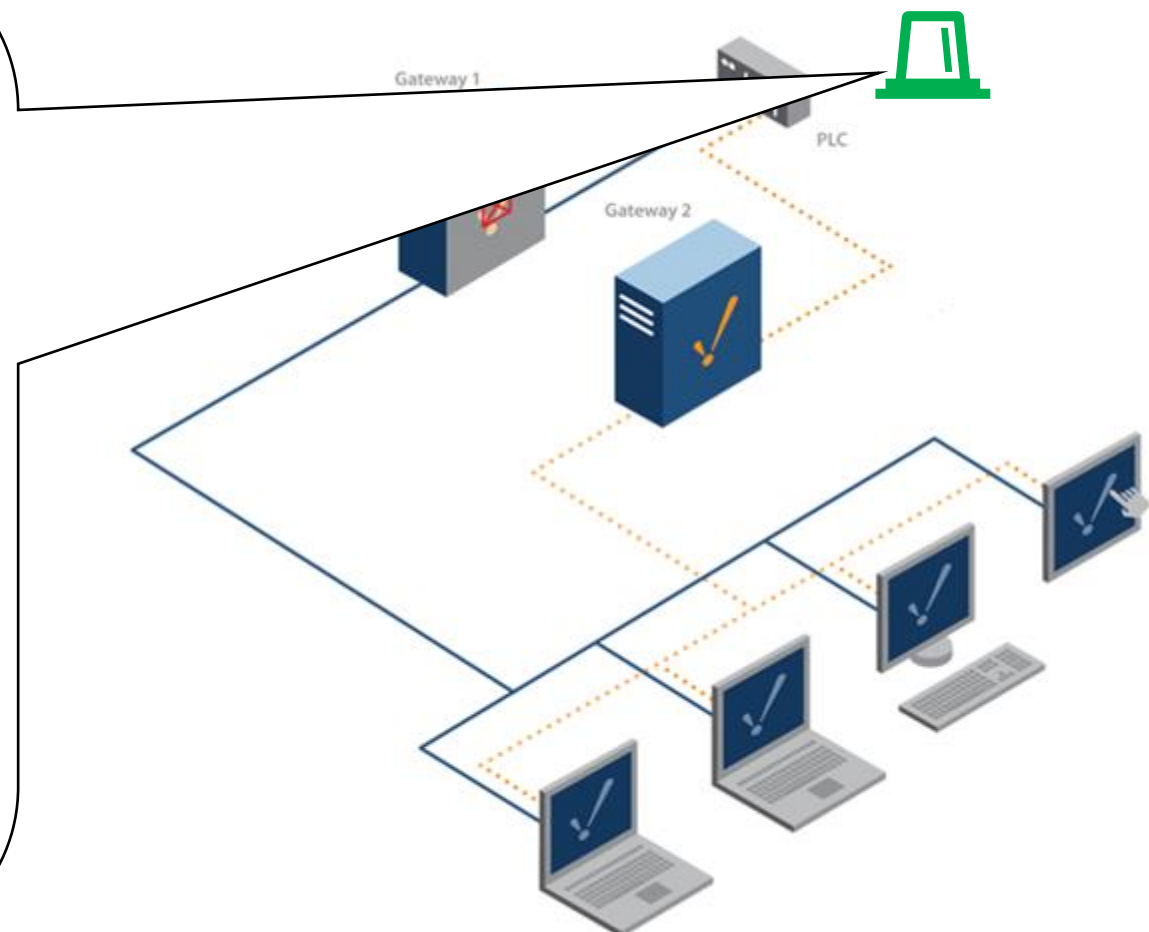
Arquitectura redundante de SCADA convencional

- Convencionalmente se tienen dos servidores.
- Uno solo está activo. Si esta falla, se active el otro.
- Ambos contienen la información actualizada.
- En la imagen hay un PLC que es un equipo que obtiene señales de la realidad y las reporta a los servidores a través de un protocolo de comunicación.
- El servidor activo recibe las señalizaciones, y las almacena en una base de datos. Se asegura de copiarle la información al otro servidor.
- Los HMI presentan la información de una forma visual e intuitiva a las personas para que observen qué está ocurriendo y puedan realizar acciones de operación/mantenimiento.



Arquitectura redundante de SCADA convencional

- Un PLC puede reportar multiples señalizaciones que pueden ser eventos, alarmas, estados de equipos, mediciones analógicas. También puede recibir un commando para operar un equipo.
- Ejemplo: Puede comunicar que un interruptor está abierto. Al recibir un commando de cierre, opera el interruptor y cuando este cierre, reporta el nuevo estado.
- Los servidores pueden estar comunicados con multiples PLCs en simultáneo.
- Cada PLC está identificado según su protocolo/medio de Comunicaciones por una IP y/o número de esclavo y/o technical key.



Arquitectura redundante de SCADA convencional

Registro de eventos

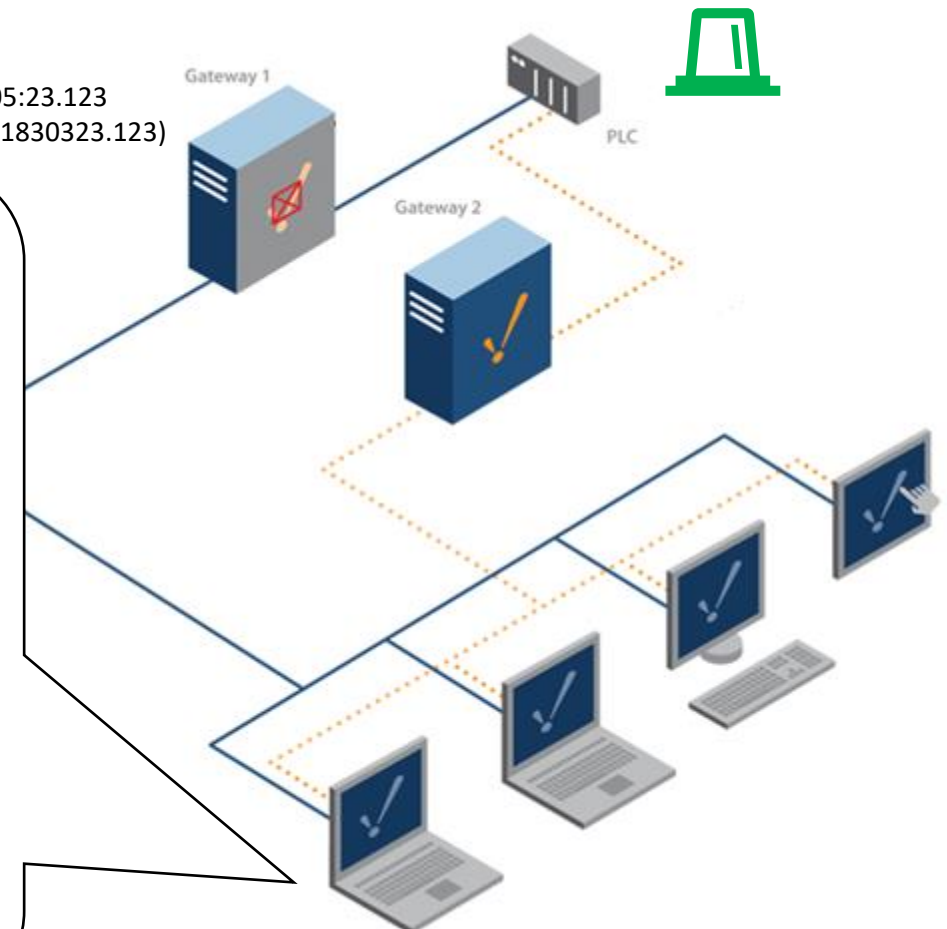
18/04/2023 15:05:23.123	Equipo ABC	Normal

Pantalla de visualización



- Value = 0 (normal)
- Quality = 0 (ok)
- TimeStamp = 18/04/2023 15:05:23.123
(UNIX format = 1681830323.123)

- Las pantallas básicas que ven las personas son las de eventos/alarmas donde observan una especie de log de lo que va ocurriendo.
- También tienen un esquema que les permite de forma intuitiva comprender el estado del Sistema.



Arquitectura redundante de SCADA convencional

Registro de eventos

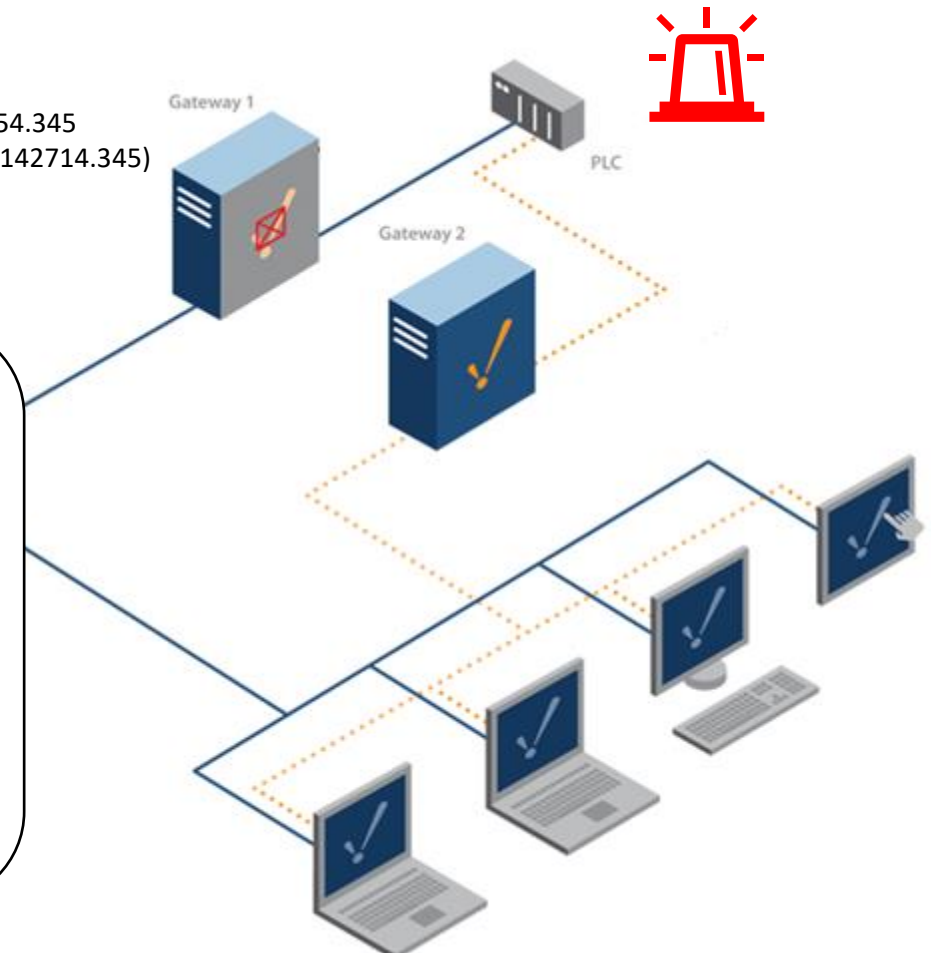
18/04/2023 15:05:23.123	Equipo ABC	Normal
22/04/2023 5:51:54.345	Equipo ABC	Alarma

Pantalla de visualización



- Value = 1 (Alarm)
- Quality = 0 (ok)
- TimeStamp = 22/04/2023 5:51:54.345
(UNIX format = 1682142714.345)

- El PLC se comunica con los servidores cuando hay un cambio de alguna señal y le informa a qué hora ocurrió.
- Esa información el HMI la representa en sus pantallas.



Registro de eventos

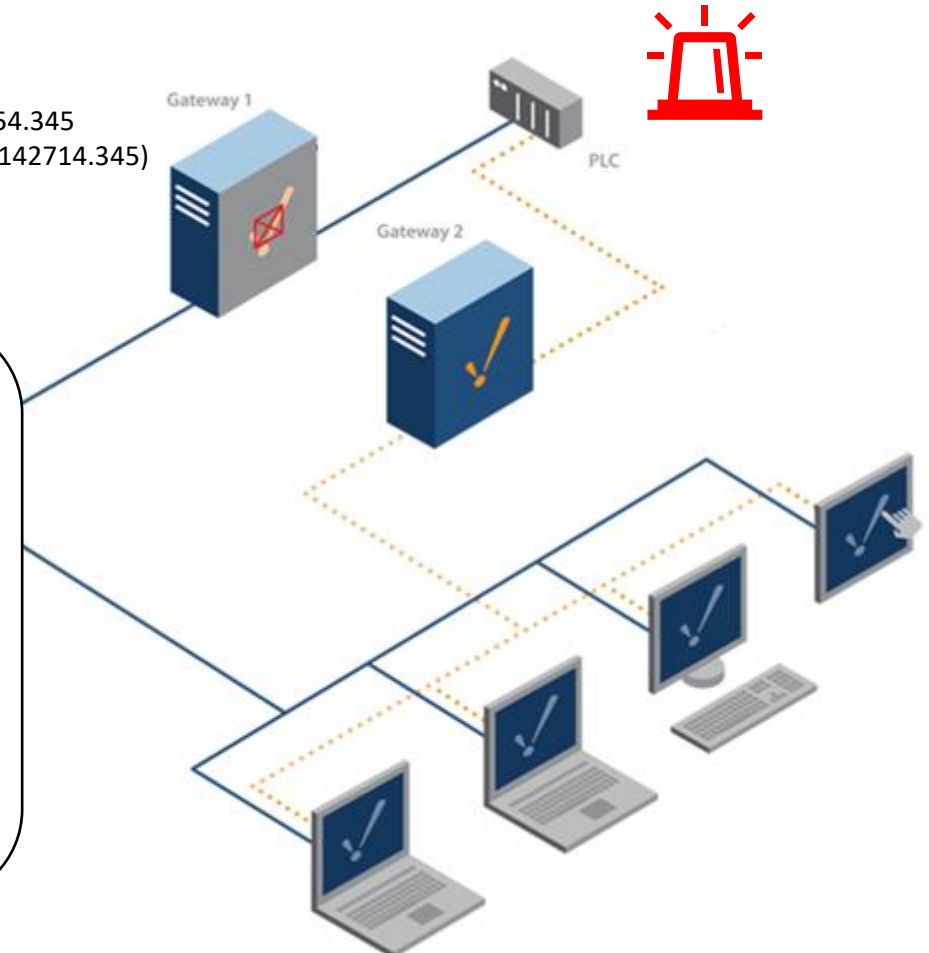
18/04/2023 15:05:23.123	Equipo ABC	Normal
22/04/2023 5:51:54.345	Equipo ABC	Alarma

Pantalla de visualización



- Value = 1 (Alarm)
- Quality = 0 (ok)
- TimeStamp = 22/04/2023 5:51:54.345
(UNIX format = 1682142714.345)

- Esa información es ordenada y presentada al usuario
- El HMI sabe que un valor 0 = ok y un 1 = Alarma



Arquitectura redundante de SCADA convencional

Registro de eventos

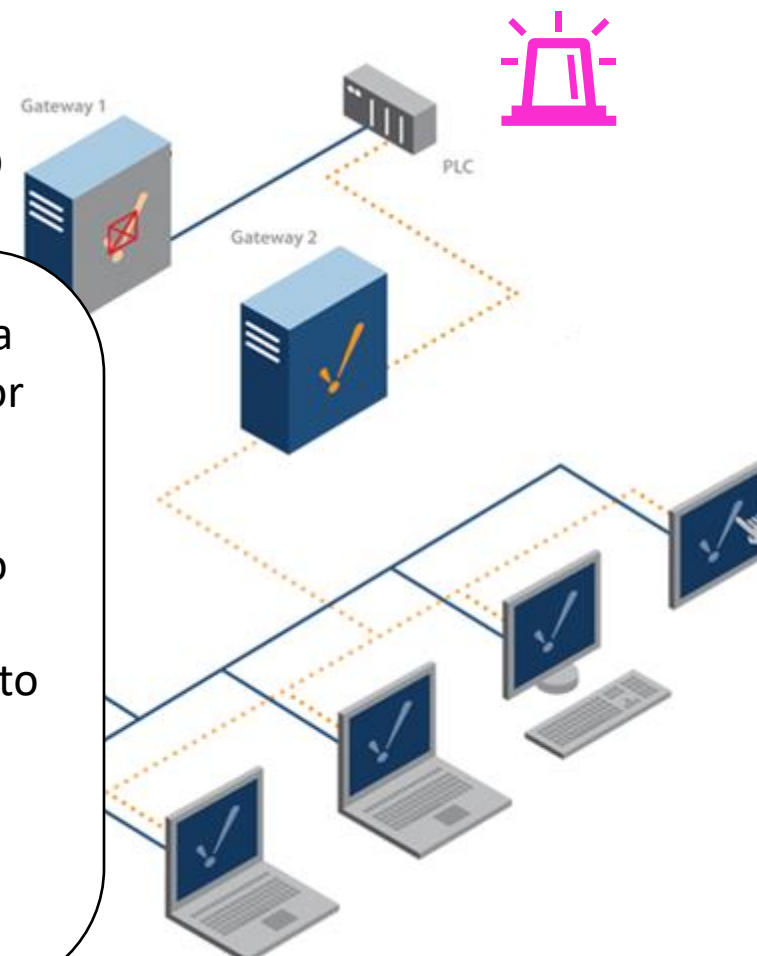
18/04/2023 15:05:23.123	Equipo ABC	Normal
22/04/2023 5:51:54.345	Equipo ABC	Alarma
22/04/2023 5:51:54.345	Equipo ABC	Desactualizado

Pantalla de visualización



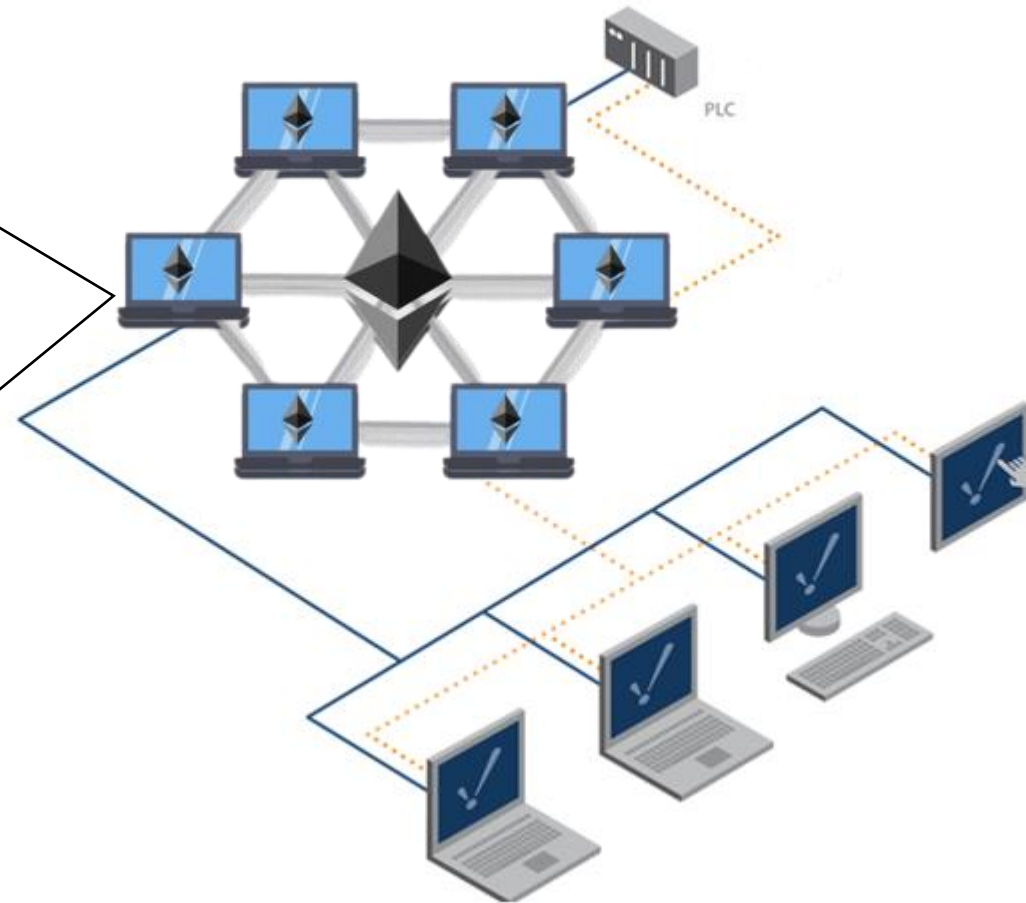
- Value = 1 (Alarm)
- Quality = 2 (Unknown – Showing last value)
- TimeStamp = 22/04/2023 5:51:54.345
(UNIX format = 1682142714.345)

- La calidad del dato se asocia a cuando el PLC informa un valor pero no está “Seguro” de que esté bien.
- Puede ser por ejemplo que no esté seguro de cómo se encuentra el equipo, por lo tanto lo reporta con calidad = 2.
- O puede ser que no sepa la hora, entonces lo indica con una calidad = 3.



Arquitectura SCADA Descentralizado

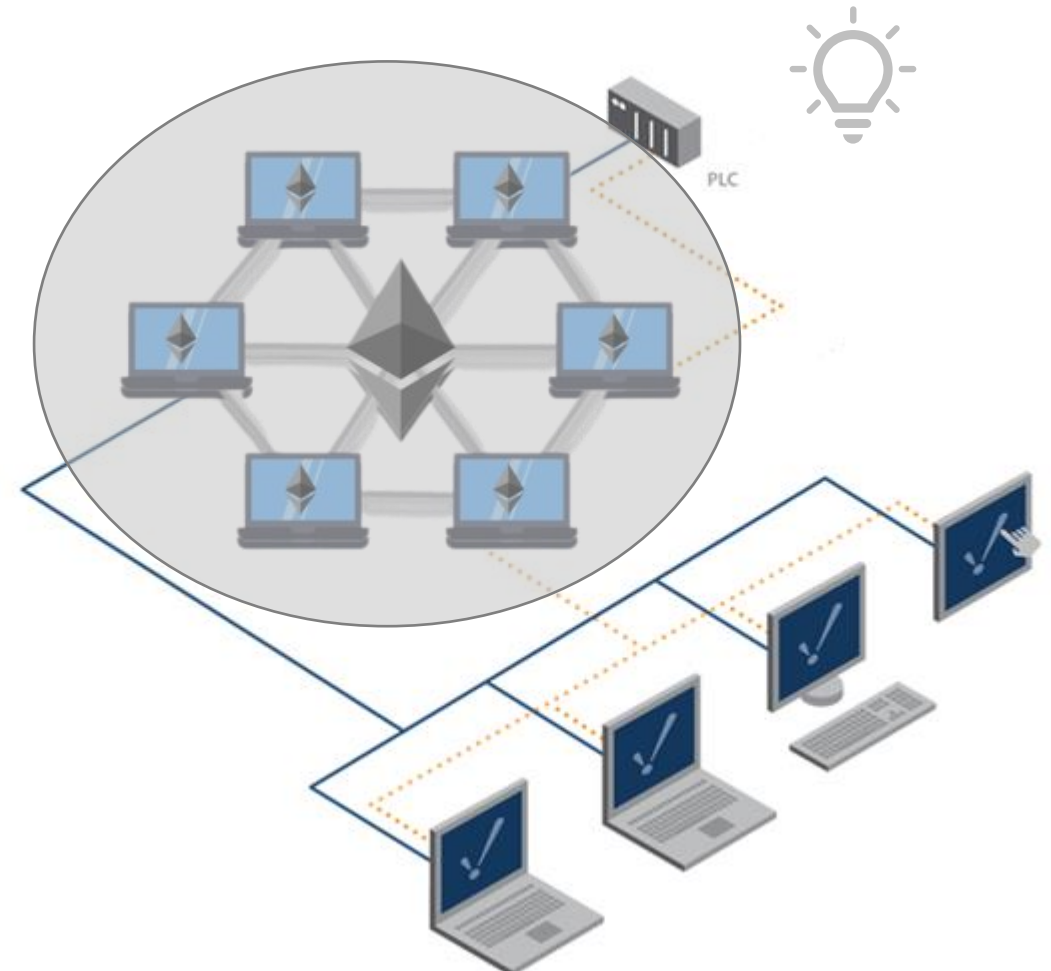
- La propuesta es descentralizar los servidores. De esa forma:
 - La base de datos y sus eventos del pasado almacenados en la blockchain serán inmutables.
 - Mejora la seguridad del al disminuir la cantidad de puntos de falla



SCADA Descentralizado – Etapa 1 ✓

Smart Contract

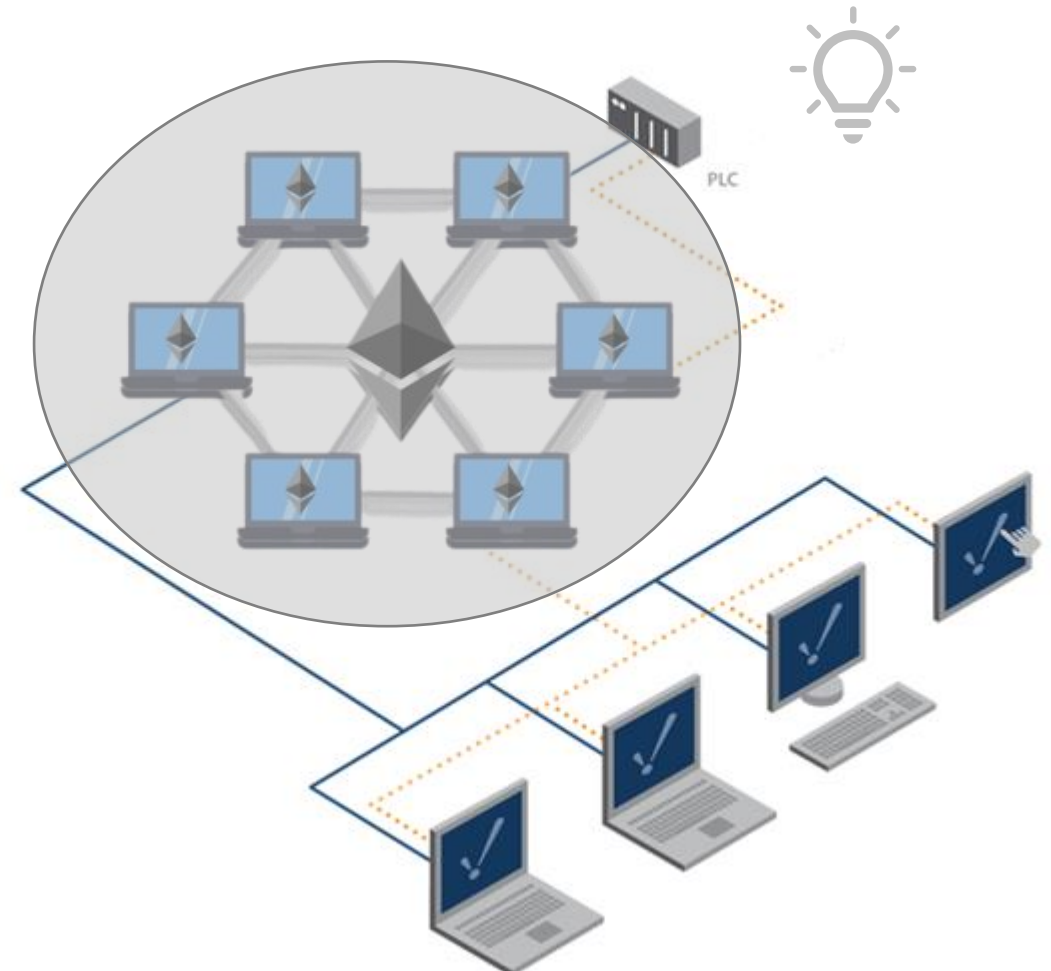
- Contiene Base de Datos. Posibles objetos:
 - Objeto *Entrada Binaria*: Valor $\in \{false, true\}$; Quality $\in (0...10)$; timestamp $\in (uint)$
 - Objeto *Comando binario*: Valor $\in \{false, true\}$; Quality $\in (0...10)$; timestamp $\in (int)$
 - Objeto entrada *Analógica*: Valor $\in (uint)$; Quality $\in (0...10)$; timestamp $\in (int)$
- Genera eventos (Librería PUSH?)



SCADA Descentralizado – Etapa 1 ✓

Smart Contract

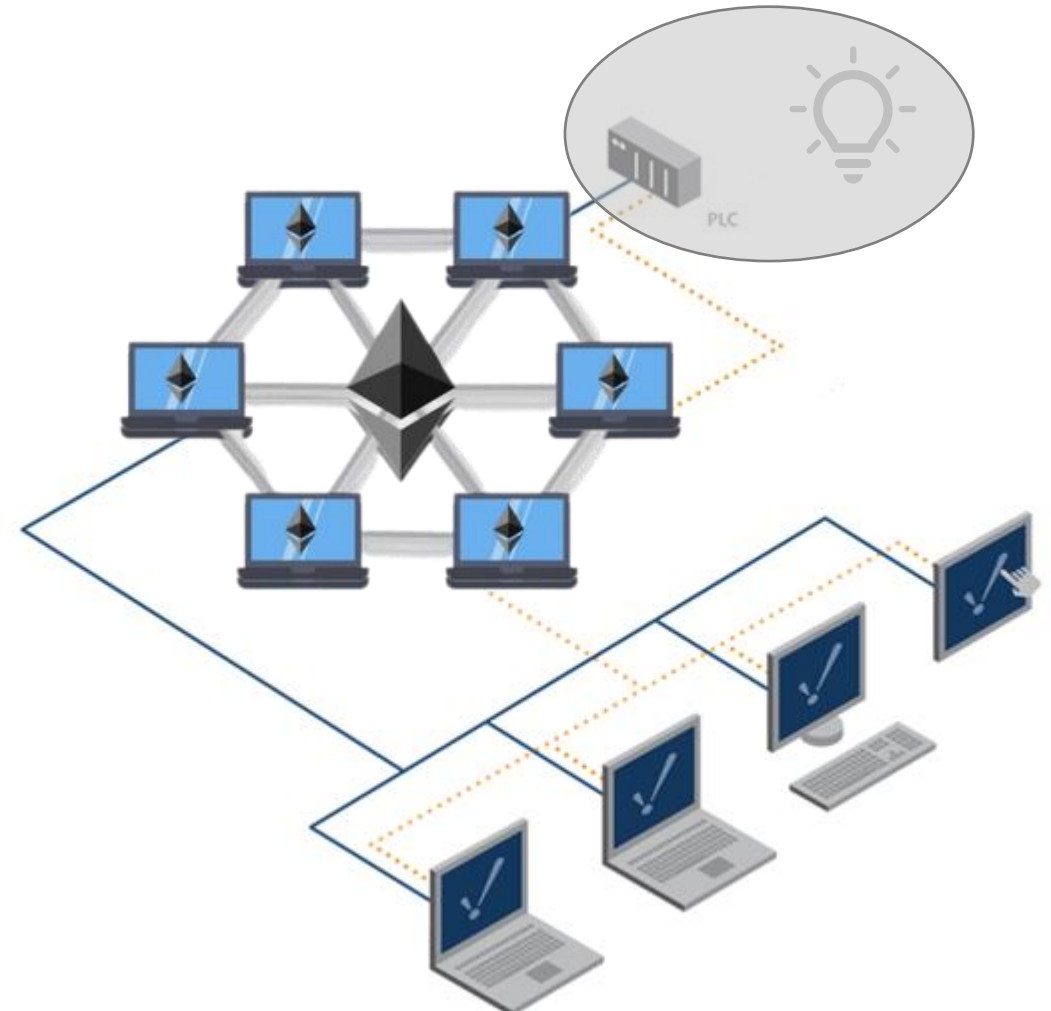
- Significados de Quality:
 - 0 = Nunca actualizado**
 - 2 = Actualizado alguna vez pero puede que haya cambiado
 - 3 = Actualizado pero la estampa de tiempo está mal
 - 10 = Correctamente actualizado**
 - 11 = Comando prosperó correctamente (solo para BO)**
- Significado de timestamp: Es un número en formato Unix Time que es un sistema de representación de la fecha y hora en segundos desde el 1 de enero de 1970 a las 00:00:00 UTC.



SCADA Descentralizado – Etapa 1 ✓

Simulador de equipo servidor de datos (esclavo)

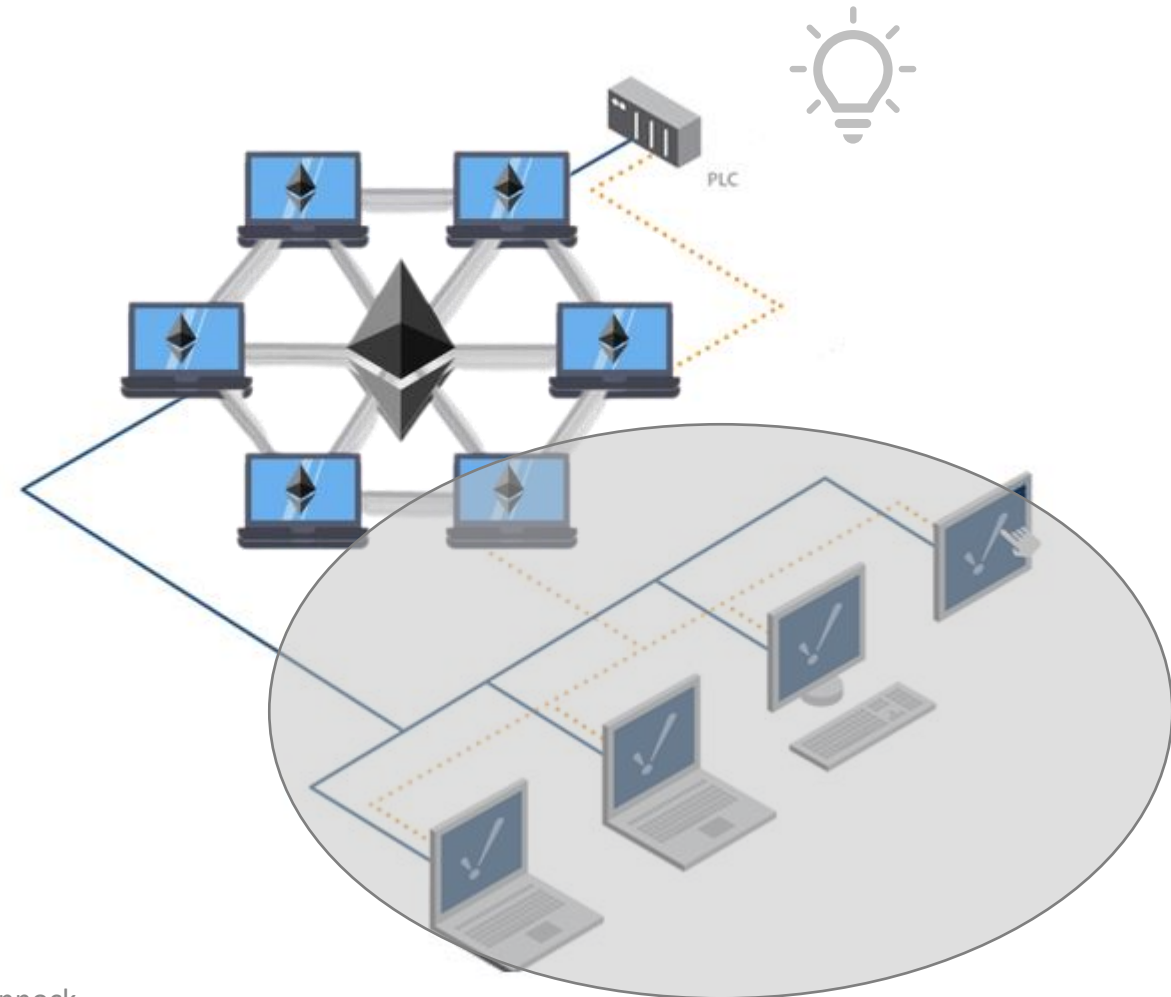
- Es una página web que nos permite simular el estado de un objeto.
- Permite forzar un valor que se escribe en la blockchain en storage o como un evento
- Permite recibir un comando y en consecuencia simular que cambia su estado
- El identificador de un esclavo sería su dirección pública



SCADA Descentralizado – Etapa 1 ✓

HMI – Interfaz gráfica

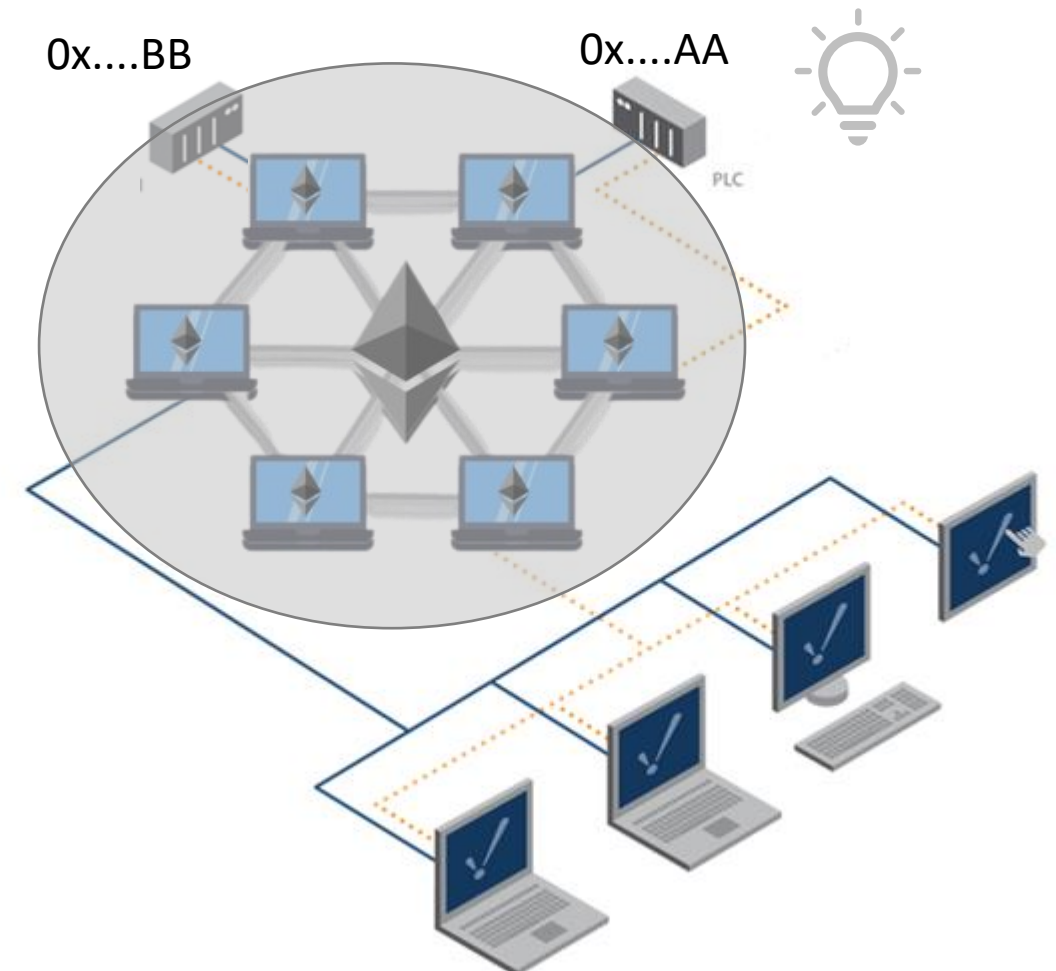
- Es una página web que nos permite ver el estado del objeto remoto leyendo variables en storage de la blockchain (o leyendo el ultimo evento?)
- Permite enviar un commando al equipo remote para cambiar su estado
- Permite ver valores analógicos
- Lleva un registro de eventos que hayan ocurrido (Lo hace a partir de los eventos de Ethereum? O lo hace reconstruyendo el historial al ir buscando las cadenas de bloques pasadas?)



SCADA Descentralizado – Etapa 1 ✓

Un PLC o cualquier equipo que reporta datos a la blockchain se llama de forma genérica **slave**. Los datos que informan los slaves se representarán en la blockchain según un array de structs con esta estructura:

```
struct slave {  
    address slave_address;  
    bool[] BI_val;  
    uint[] BI_q;  
    uint[] BI_t;  
    int[] AI_val;  
    uint[] AI_q;  
    uint[] AI_t;  
    bool[] BO_val;  
    uint[] BO_q;  
    uint[] BO_t;  
}
```



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 5);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

Imaginemos que tenemos un slave con 3 entradas binarias, 2 salidas binarias y 5 entradas analógicas.

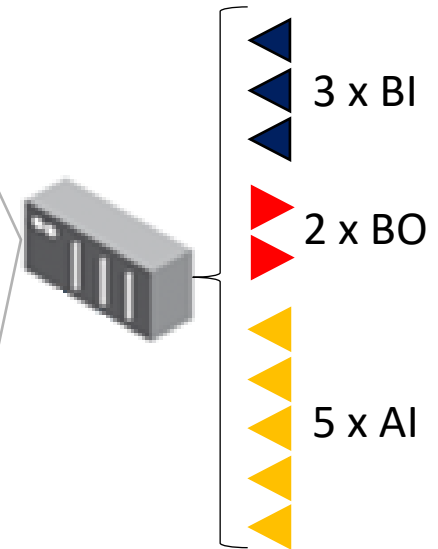
Se tratan de conexiones eléctricas las cuáles el PLC las vincula de forma digital hacia la blockchain en nuestro caso.

Una entrada Binaria (BI) puede recibir estados, por ejemplo de una válvula abierta (=true) o cerrada (=false).

Una salida binaria abrir o cerrar contactos eléctricos, por ejemplo para abrir una válvula (=false) o cerrarla (=true).

Una entrada analógica (AI) puede recibir por ejemplo una temperatura a través de un transductor que convierte °C en Volts.

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

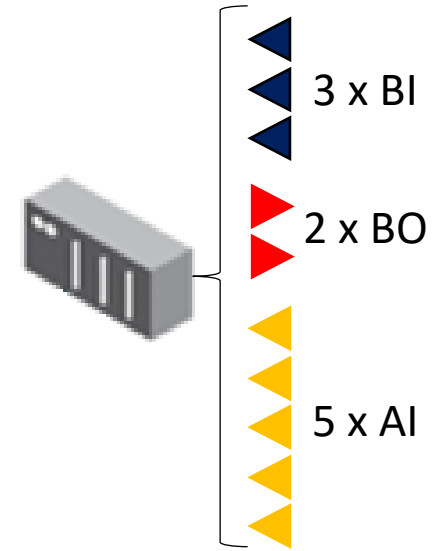
// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 5);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

Esta sección de código sirve para hacer el deploy del contrato de forma local con hardhat.

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

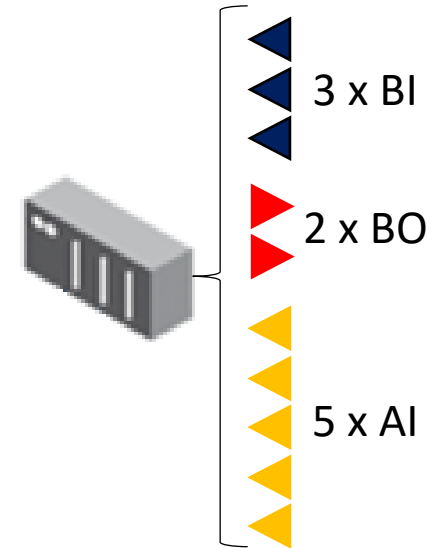
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 5);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

La función **CreateSlave** crea un slave en la base de datos, indicando como input su address. Por ser el primer slave se corresponderá con el índice 0.

```
slave({
  slave_address: 0x...AA,
  BI_val: [],
  BI_q: [],
  BI_t: [],
  AI_Val: [],
  AI_q: [],
  AI_t: [],
  BO_Val: [],
  BO_q: [],
  BO_t: []
});
```

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

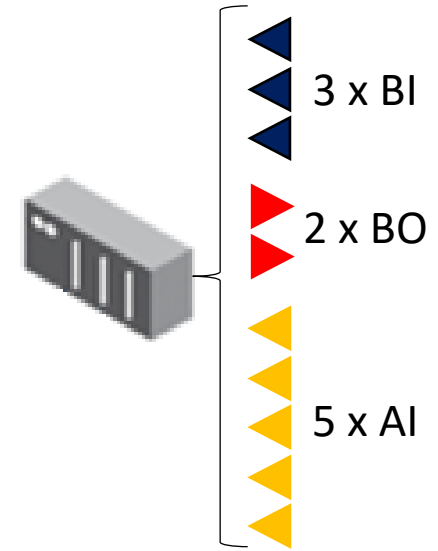
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 5);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

La función `CreateBinaryInputs` crea 3 BIs.

```
slave({
  slave_address: 0x...AA,
  BI_val: [0,0,0],
  BI_q: [0,0,0],
  BI_t: [0,0,0],
  AI_Val: [],
  AI_q: [],
  AI_t: [],
  BO_Val: [],
  BO_q: [],
  BO_t: []
});
```

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

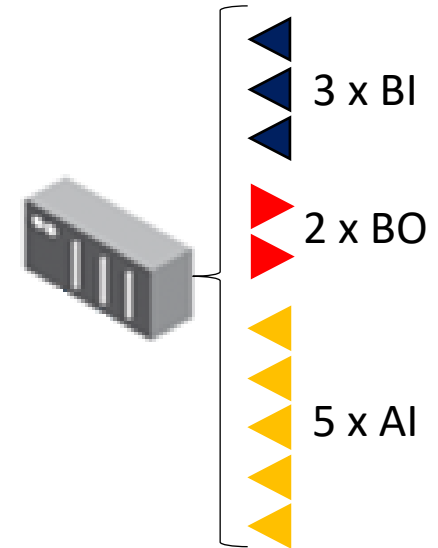
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 3);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

La función `CreateAnalogInputs` crea 3 AIs.

```
slave({
  slave_address: 0x...AA,
  BI_val: [0,0,0],
  BI_q: [0,0,0],
  BI_t: [0,0,0],
  AI_Val: [0,0,0],
  AI_q: [0,0,0],
  AI_t: [0,0,0],
  BO_Val: [],
  BO_q: [],
  BO_t: []
});
```

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

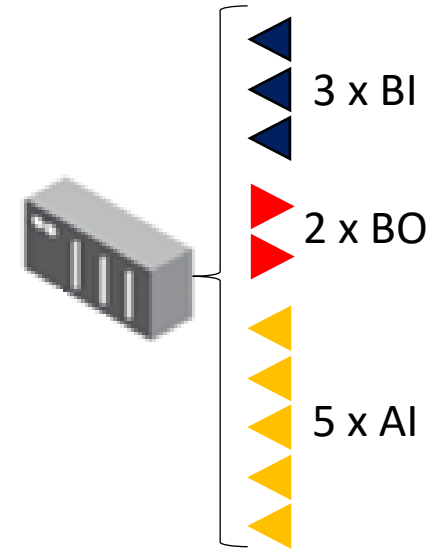
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 3);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

La función `CreateBinaryOutputs` crea 2 BOs.

```
slave({
  slave_address: 0x...AA,
  BI_val: [0,0,0],
  BI_q: [0,0,0],
  BI_t: [0,0,0],
  AI_Val: [0,0,0],
  AI_q: [0,0,0],
  AI_t: [0,0,0],
  BO_Val: [0,0],
  BO_q: [0,0],
  BO_t: [0,0]
});
```

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

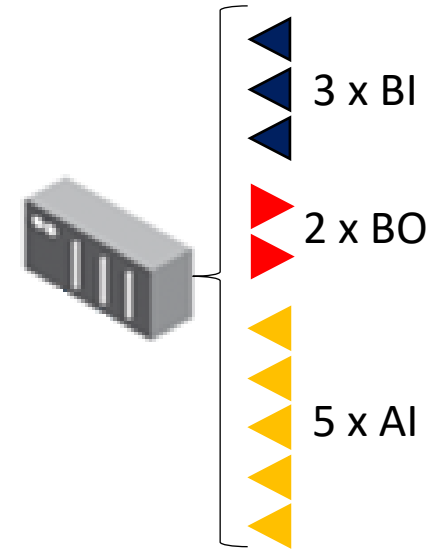
// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 3);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

Esta linea en hardhat sirve para simular que el PLC se conecta al Smart Contract desplegado.

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

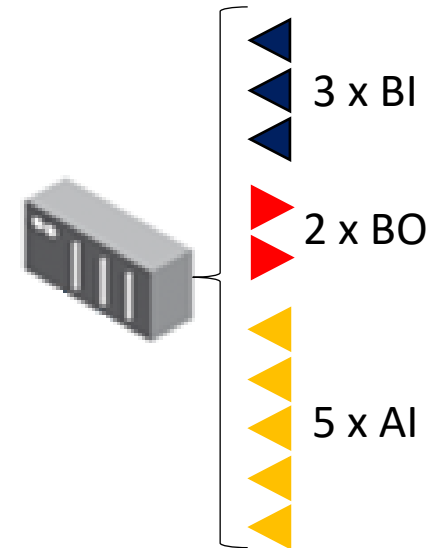
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 3);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

Aquí se arman todos los datos que este PLC simulado enviará a la blockchain para actualizar la base de datos.

En un caso real, es posible que el PLC no envíe inmediatamente los cambios. Por eso es que siempre los envía con estampa de tiempo.

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

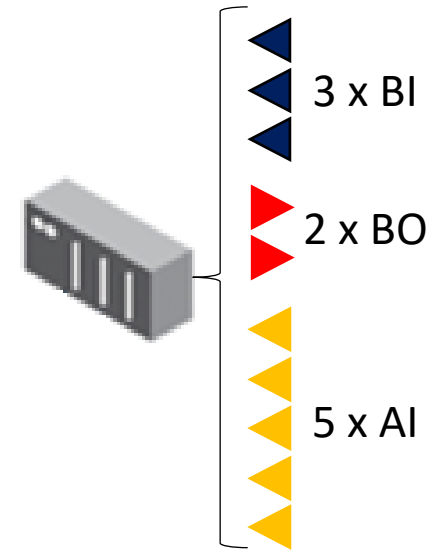
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 5);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

La función **setProcessObjects** enviada por el PLC actualiza los datos en la Base de Datos.

```
slave({
  slave_address: 0x...AA,
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251,
1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251,
1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
});
```

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
// Local deploy with hardhat
const SCADA = await ethers.getContractFactory("SCADA");
const hardhatSCADA = await SCADA.deploy();
await hardhatSCADA.deployed();

// get the addresses. owner should be the address of a root or engineering account. slave 1 corresponds to the address of a slave.
const [owner, slave1, HMI] = await ethers.getSigners();

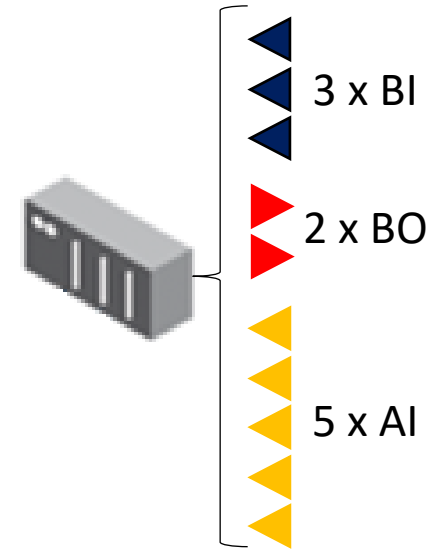
await hardhatSCADA.CreateSlave(slave1.address);
await hardhatSCADA.CreateBinaryInputs(slave1.address, 3);
await hardhatSCADA.CreateAnalogInputs(slave1.address, 5);
await hardhatSCADA.CreateBinaryOutputs(slave1.address, 2);

// Connect as slave1
const slave1ContractConnection = hardhatSCADA.connect(slave1);
// Define process object
const processObjects = {
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
};
// Call function as slave1 to set all the process objects
await slave1ContractConnection.setProcessObjects(
  processObjects.BI_val,
  processObjects.BI_q,
  processObjects.BI_t,
  processObjects.AI_Val,
  processObjects.AI_q,
  processObjects.AI_t,
  processObjects.BO_Val,
  processObjects.BO_q,
  processObjects.BO_t
);
```

La función `setProcessObjects` enviada por el PLC actualiza los datos en la Base de Datos.

```
slave({
  slave_address: 0x...AA,
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251,
1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251,
1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
});
```

Address: 0x....AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
slave({  
  slave_address: 0x...AA,  
  BI_val: [true, false, true],  
  BI_q: [10, 10, 10],  
  BI_t: [1683258250, 1683258251, 1683258252],  
  AI_Val: [12345, 0, -1314],  
  AI_q: [10, 10, 10],  
  AI_t: [1683258250, 1683258251, 1683258252],  
  BO_Val: [false, false],  
  BO_q: [0, 0],  
  BO_t: [0, 0]  
});
```

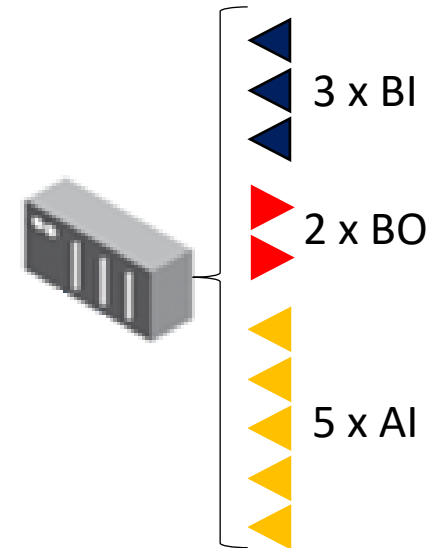
Por ejemplo, aquí la primera entrada binaria (la número 0) está en true. Podría significar que una válvula está cerrada.

La calidad es 10, lo que significa que está correctamente actualizada.

La timestamp es 1683258250 que significa que su último estado fue actualizado en este momento:

May 05 2023 03:44:10

Address: 0x....AA



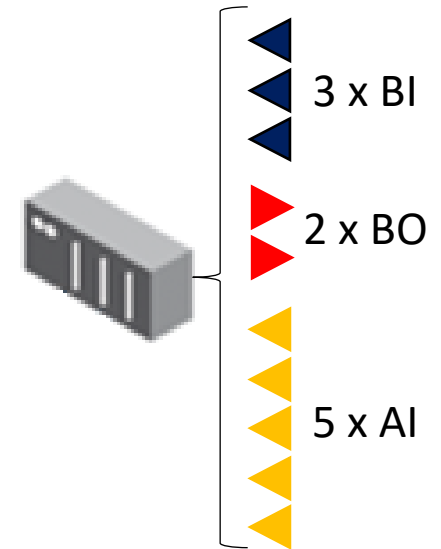
SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
slave({
  slave_address: 0x...AA,
  BI_val: [true, false, true],
  BI_q: [3, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
});
```

Si la calidad fuese de 3, significaría que su valor es true, pero que el PLC no tenía su reloj interno sincronizado, por lo tanto es su forma de avisar que la estampa de tiempo indicada puede estar mal.

Address: 0x...AA



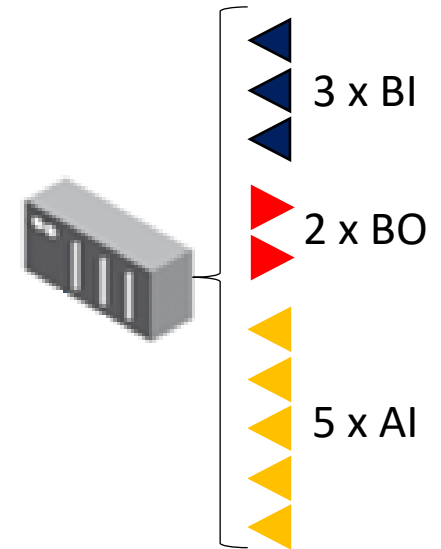
SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
slave({  
  slave_address: 0x...AA,  
  BI_val: [true, false, true],  
  BI_q: [10, 10, 10],  
  BI_t: [1683258250, 1683258251, 1683258252],  
  AI_Val: [12345, 0, -1314],  
  AI_q: [10, 10, 10],  
  AI_t: [1683258250, 1683258251, 1683258252],  
  BO_Val: [false, false],  
  BO_q: [0, 0],  
  BO_t: [0, 0]  
});
```

Aquí hay 3 entradas analógicas cuyos valores eran los indicados a la hora indicada en la estampa de tiempo.

Address: 0x...AA



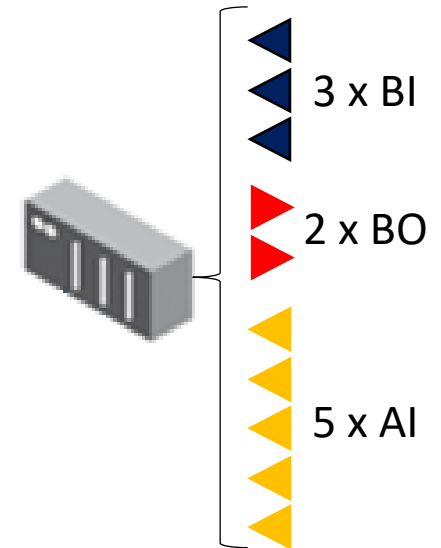
SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat

```
slave({  
  slave_address: 0x...AA,  
  BI_val: [true, false, true],  
  BI_q: [10, 10, 10],  
  BI_t: [1683258250, 1683258251, 1683258252],  
  AI_Val: [12345, 0, -1314],  
  AI_q: [10, 10, 10],  
  AI_t: [1683258250, 1683258251, 1683258252],  
  BO_Val: [false, false],  
  BO_q: [0, 0],  
  BO_t: [0, 0]  
});
```

En este ejemplo, nunca fue emitido ningún comando. Por eso la calidad es 0 y la estampa de tiempo también.

Address: 0x...AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat - Continuación

```
// Connect as HMI
const HMIContractConnection = hardhatSCADA.connect(HMI);
// Read everything
const DB = await HMIContractConnection.ReadDB();

await expect(HMIContractConnection.SetBO(slave1.address,1,true,0,1683258300))
DB = await HMIContractConnection.ReadDB();
```

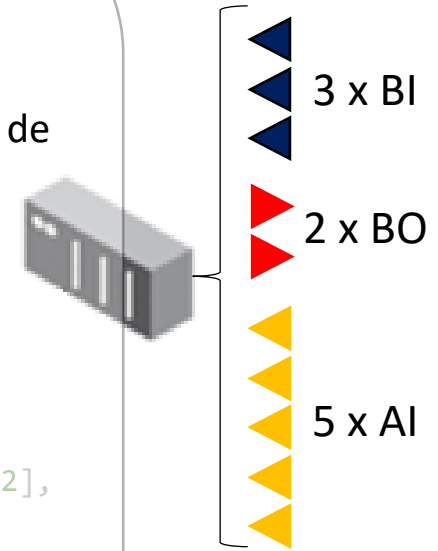
Ahora simulamos que nos conectamos con la dirección del HMI.

La función `ReadDB` leerá todo el array de structs de slaves.

Es decir, toda la base de datos completa.

```
slave({
  slave_address: 0x...AA,
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, false],
  BO_q: [0, 0],
  BO_t: [0, 0]
});
```

Address: 0x...AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat - Continuación

```
// Connect as HMI
const HMIContractConnection = hardhatSCADA.connect(HMI);
// Read everything
const DB = await HMIContractConnection.ReadDB();

await expect(HMIContractConnection.SetBO(slave1.address,1,true,10,1683258300))
DB = await HMIContractConnection.ReadDB();
```

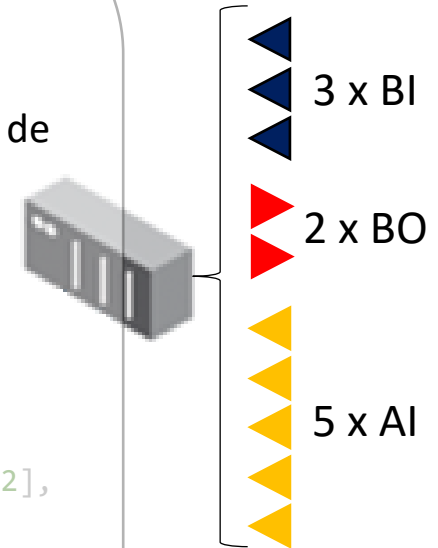
Ahora simulamos que nos conectamos con la dirección del HMI.

La función `ReadDB` leerá todo el array de structs de slaves.

Es decir, toda la base de datos completa.

```
slave({
  slave_address: 0x...AA,
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, true],
  BO_q: [0, 10],
  BO_t: [0, 1683258300]
});
```

Address: 0x...AA



SCADA Descentralizado – Etapa 1 ✓

Ejemplo de Código de test en hardhat - Continuación

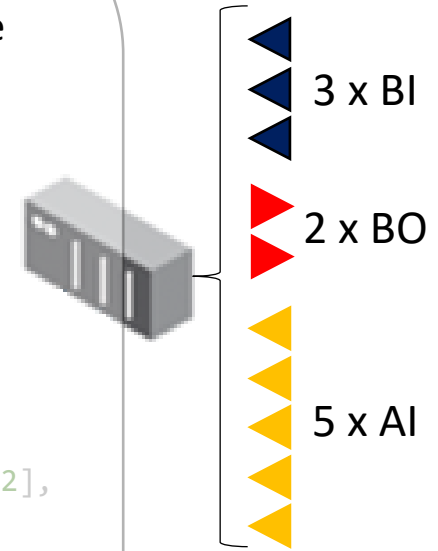
// Connect as HMI

```
DB = await HMIContractConnection.ReadDB();
```

Un PLC real debería recibir el comando, y en caso de ser exitoso, actualizar la estampa de tiempo e indicar con un 11 que el comando prosperó correctamente. El HMI podrá leer esta actualización en la base de datos y sabrá que el comando fue correcto.

```
slave({
  slave_address: 0x...AA,
  BI_val: [true, false, true],
  BI_q: [10, 10, 10],
  BI_t: [1683258250, 1683258251, 1683258252],
  AI_Val: [12345, 0, -1314],
  AI_q: [10, 10, 10],
  AI_t: [1683258250, 1683258251, 1683258252],
  BO_Val: [false, true],
  BO_q: [0, 11],
  BO_t: [0, 1683258500]
});
```

Address: 0x...AA



SCADA Descentralizado – Etapa 2.1

DAPP PLC

Se trata de una página web que simula ser un PLC que se conecta a la blockchain.

La página puede mostrar por ejemplo:

- Una valvula que puede estar abierta o cerrada. Con un botón se podría cambiar su estado.
- Una alarma que puede estar normal o alarma.
- Un cuadro de texto donde se escribe una temperatura.
- Un botón con el cuál al apretarlo se escriban los últimos datos en la blockchain.

La DAPP debería poder “recibir comandos”. Una forma de implementar esto es que periódicamente verifique si hay algún cambio en algún Binary Output.

Pero debería haber algún método más eficiente, como recibir un “evento” o algo así. De esa forma cuando un HMI escribe un comando, se genera un evento a partir del cuál el PLC se entera de que se le requiere un commando. -> **Investigar**

Al recibir un commando por ejemplo de abrir, debería similar que abre la valvula. Y devolver a la BI correspondiente el cambio de estado con la estampa de tiempo, y también escribir un 11 en la calidad del BO con la estampa de tiempo actualizada.

SCADA Descentralizado – Etapa 2.2

DAPP SCADA/HMI – Pantallas de operación

Se trata de una página web que muestra la visualización de la base de datos.

La página puede mostrar por ejemplo lo mismo que el PLC. La diferencia es que los dibujos cambiarían en función de lo que está escrito en la base de datos.

- Una valvula que puede estar abierta o cerrada. Con un botón se podría enviar un comando hacia el PLC para cambiar el estado.
- Una alarma que puede estar normal o alarma.
- Un cuadro de texto donde se muestra una temperatura.
- Un botón con el cuál al apretarlo se fuerce una lectura de la base de datos y se actualicen los dibujos.

La DAPP podría verificar periódicamente verifique si hay algún cambio en la base de datos para actualizar las pantallas. Pero debería haber algún método más eficiente, como recibir un “evento” o algo así. De esa forma cuando un PLC realiza un cambio, se genera un evento a partir del cuál el HMI se entera que debe actualizar su pantalla. -> **Investigar**

SCADA Descentralizado – Etapa 2.2

DAPP SCADA/HMI – Pantallas de operación

Cómo se representan los valores de calidad. Ejemplo si se trata del estado de una valvula:

0: Significa que nunca se actualize ese valor, por lo tanto se desconoce su estado. Dibujar un signo de pregunta ?.

10: Su valor es correcto. Por lo tanto el dibujo dependerá de si la valvula está abierta o cerrada según el BI_Val.

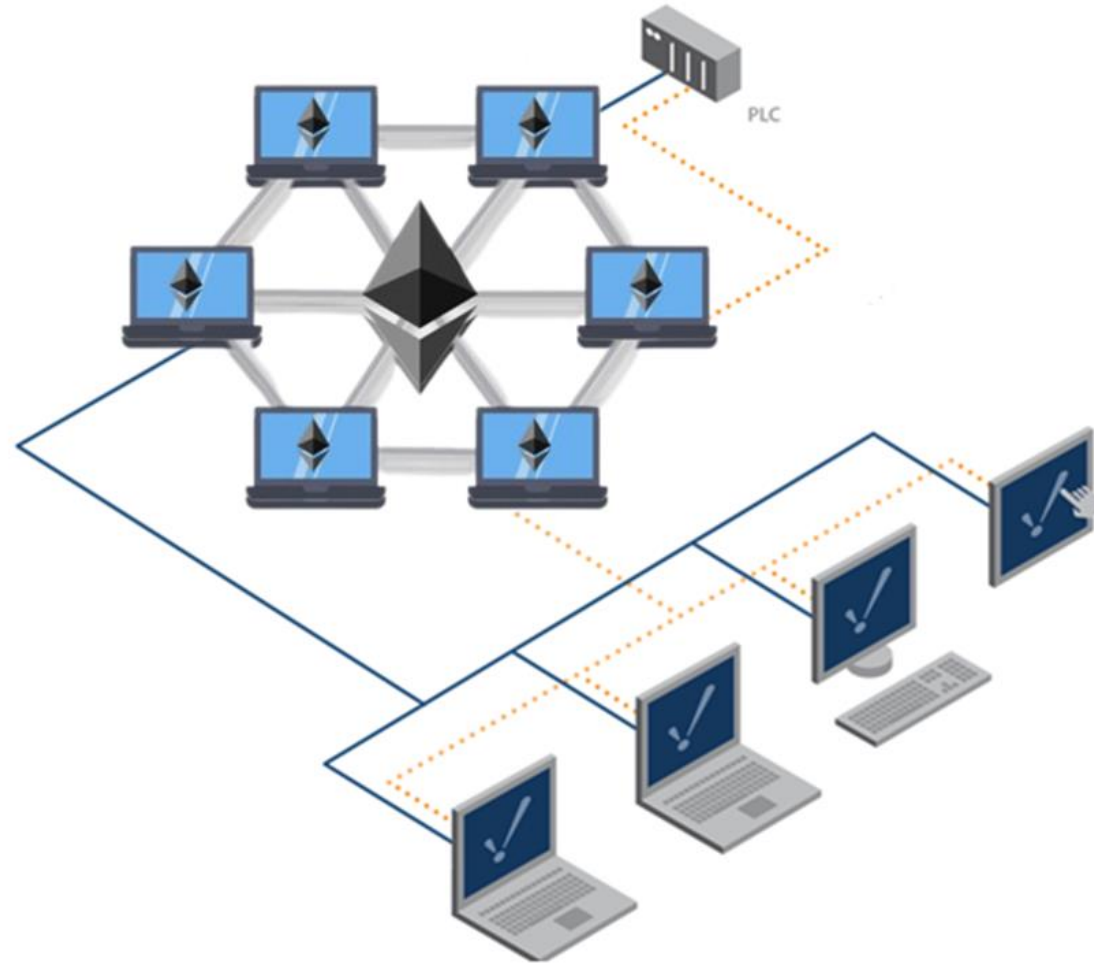
Otros valores. Tal vez requiera un esfuerzo que en esta etapa no vale la pena.

2: El dibujo dependerá de si la valvula está abierta o cerrada según el BI_Val. Pero debe estar pintado de magenta, lo que significa que solo se muestra el ultimo estado pero es posible que haya cambiado.

3: El dibujo dependerá de si la valvula está abierta o cerrada según el BI_Val. Debe estar pintado de Amarillo, lo que significa que su estado es correcto, pero el PLC avisa que la estampa de tiempo no es correcta.

Etapas futuras

Cualquiera de las próximas tareas puede realizarse de forma independiente y agregar funcionalidades y mejoras



SCADA Descentralizado – Etapa 3

DAPP SCADA/HMI – Pantallas de eventos

Se trata de un log que muestre todo lo que fue ocurriendo en la historia del smart contract:




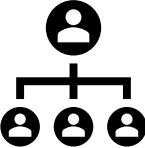
La forma de realizarlo es recorrer los bloques hacia el pasado verificando su base de datos e imprimiendo cómo estaba la base de datos en cada momento. Los logs podrían ser básicamente:
estados/alarmas y sus cambios con la estampa de tiempo asociada
Qué dirección envió qué comando y hacia qué PLC

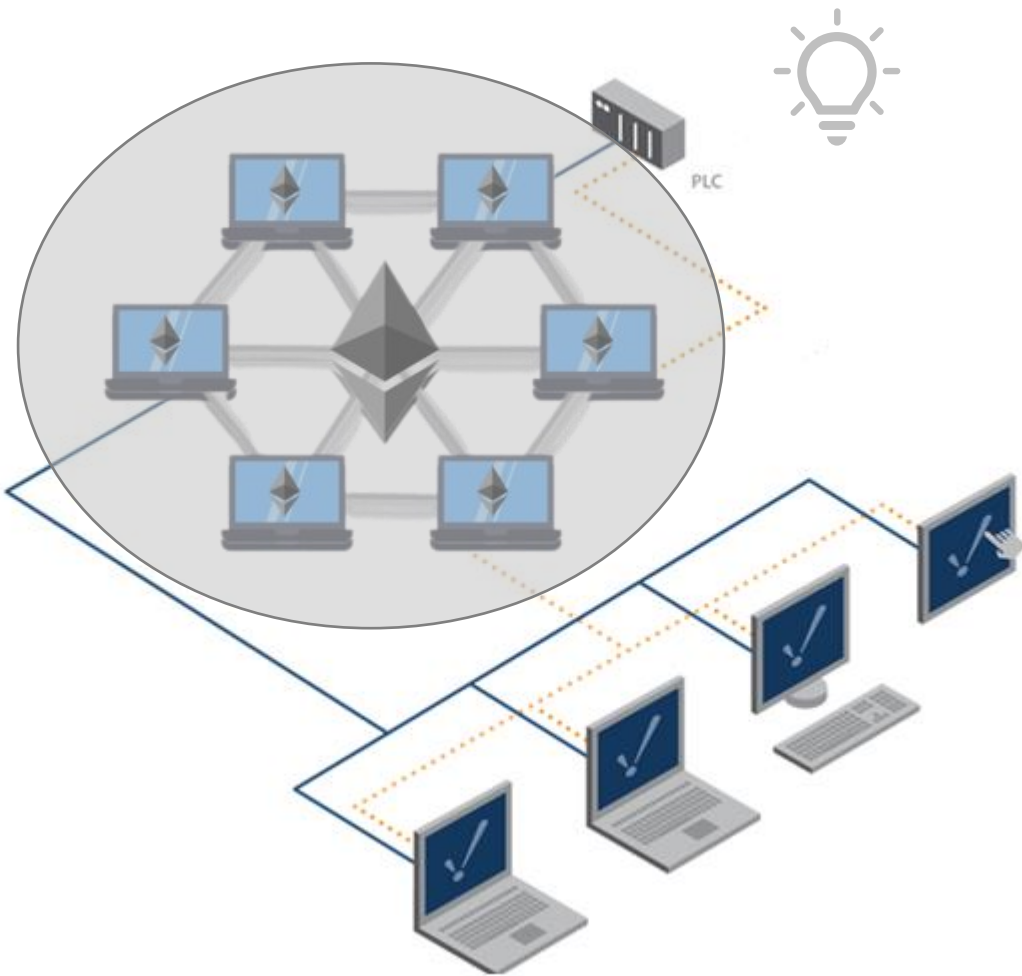
También podría hacerse un histórico de las analógicas, pero es un gran esfuerzo que no vale la pena en esta etapa.

SCADA Descentralizado – Etapa 4

Smart Contract – Permisos y usuarios

- Contiene usuarios y maneja permisos (Librería openzeppelin?)

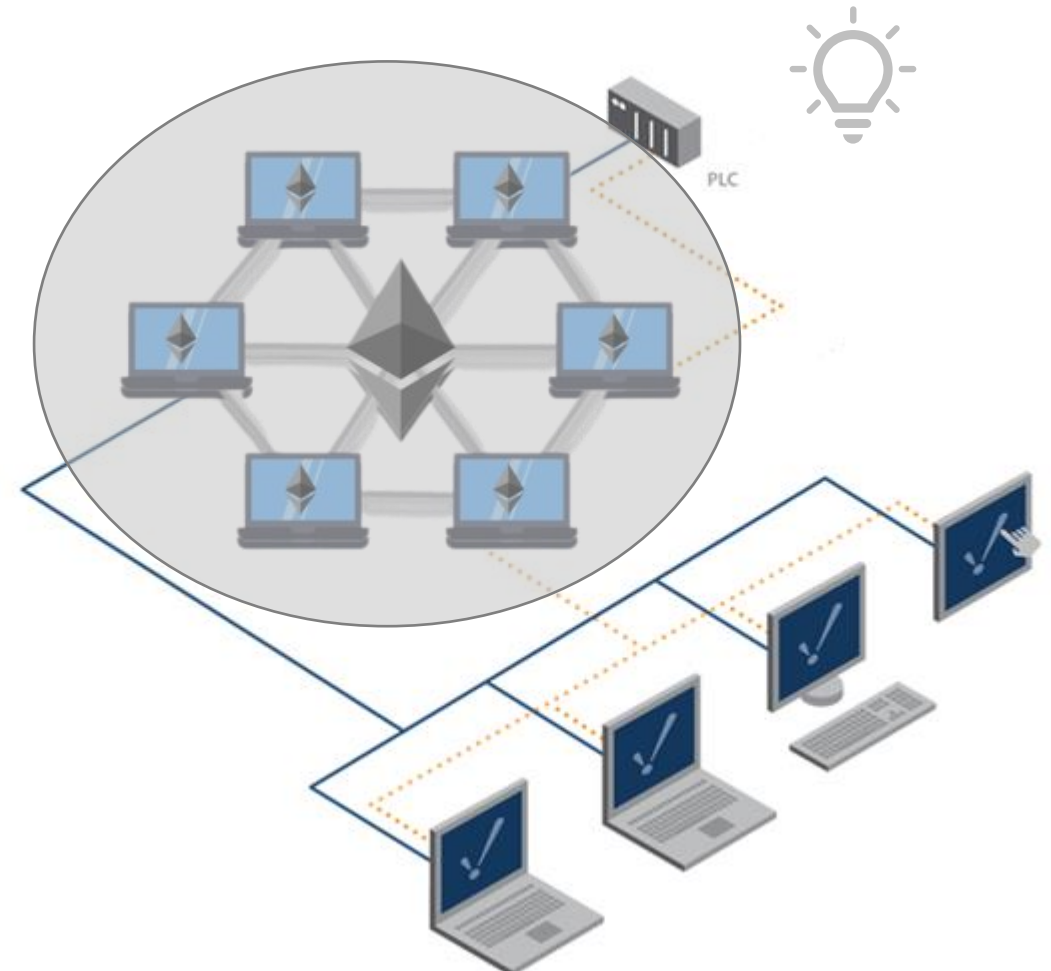
	Visualización: Solo puede ver. Todos pueden ver.
	Operación: Puede enviar comandos. Su dir. pública estará guardada en el SC.
	Ingeniería: Puede agregar-eliminar objetos-esclavos en el SC. Su dir. pública estará guardada en el SC.
	Root: Puede agregar-eliminar-modificar usuarios y permisos. Su dir. pública estará guardada en el SC.



SCADA Descentralizado – Etapa 5

Smart Contract – Supervisión de equipos de comunicación

- El Smart Contract automáticamente podría “supervisar” al PLC. Eso significa que periódicamente verifica una conexión con el PLC. En caso de no tener respuesta, puede colocar en alarma en su base de datos.
- Los HMI recibirán ese estado y sabrán que perdieron conexión con ese equipo.
- La periodicidad de esta verificación puede ser modificada por un usuario de ingeniería



SCADA Descentralizado – Etapa 6

HMI – Interfaz gráfica y usuario de ingeniería

- Permite fácilmente realizar tareas de ingeniería para insertar objetos en la pantalla de operación/visualización y crear también los objetos en el SC
- Permite fácilmente que el root visualice y modifique permisos de usuarios

