

FAWAZ MALIK 101461582

NAMIT MANI 101383808

ANIKET SRIVASTRA 101469899

BADARUDDIN KHUHRO 101467663

HEMANTH KUMAR Pothuri 101464127

BCDV 1025 – Enterprise Blockchain Development

FINAL GROUP PROJECT

Terrarium IoT Hyperledger project integration with Front end

Summary

This a terrarium IoT (internet of Things) project, where Hyperledger fabric blockchain is being implemented on the back end for user data and pet data is being stored on the blockchain as user buys the pet and the tank. Once tank is purchased by the customer from shopowner, the shopowner provides tank setup services for the end customer. The shopowner inventory is updated and manufacturer has received a transaction signature.

Business Use Case

Our blockchain services cater to shop owners and terrarium manufacturers, utilizing Hyperledger Fabric for seamless business operations. A simple use case involves a purchase transaction between a shop owner and a customer. Through the blockchain, the customer can securely place an order for a terrarium and make the payment. The transaction details are recorded on the blockchain, providing transparency and trust. The manufacturer receives the order information and prepares the terrarium for shipment. The blockchain enables efficient communication between the shop owner and manufacturer, ensuring a smooth purchasing process while enhancing the customer's experience.

Solutions implementation Design

This section covers the solution implementation steps, technologies and platforms undertaken to deploy the project. Our main development and deployment tools and technologies include the following

- Hyperledger Fabric Blockchain
- Visual Studio
- Microsoft AZURE virtual machines
- ERN (Express.js, React.js, Node.js) application STACK

These sub sections are covered below

AZURE:

On this platform, a Linux virtual machine is being created where Hyperledger fabric blockchain would be deployed.


Step 1: Here we create a virtual machine with selecting parameters best fit required for the project

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ	<div>Azure for Students</div>
Resource group * ⓘ	<div>(New) Resource group</div> <div>Create new</div>

Instance details

Virtual machine name * ⓘ	<div></div>
Region * ⓘ	<div>(Canada) Canada Central</div>
Availability options ⓘ	<div>Availability zone</div>
Availability zone * ⓘ	<div>Zones 1</div>
<div> You can now select multiple zones. Selecting multiple zones will create one VM per zone. Learn more</div>	
Security type ⓘ	<div>Trusted launch virtual machines</div>

[Review + create](#)


[< Previous](#)

[Next : Disks >](#)

Step 2: Selecting ssh connection authentication to connect to the vm remotely

VM architecture ⓘ	<div><div><input type="radio"/> Arm64</div><div><input checked="" type="radio"/> x64</div></div>
Run with Azure Spot discount ⓘ	<div><input type="checkbox"/></div>
Size * ⓘ	<div>Standard_D2s_v3 - 2 vcpus, 8 GiB memory (\$81.03/month)</div> <div>See all sizes</div>

Administrator account

Authentication type ⓘ	<div><div><input checked="" type="radio"/> SSH public key</div><div><input type="radio"/> Password</div></div>
<div><div> Azure now automatically generates an SSH key pair for you and allows you to store it for future use. It is a fast, simple, and secure way to connect to your virtual machine.</div></div>	
Username * ⓘ	<div>azureuser</div>
SSH public key source	<div>Generate new key pair</div>
Key pair name *	<div>Name the SSH public key</div>

[Review + create](#)

[< Previous](#)

[Next : Disks >](#)

Step 3 : Create and attach virtual disk to the vm

VM disk encryption

Azure disk storage encryption automatically encrypts your data stored on Azure managed disks (OS and data disks) at rest by default when persisting it to the cloud.

Encryption at host ⓘ

☐

i Encryption at host is not registered for the selected subscription.
[Learn more about enabling this feature](#) ↗

OS disk

OS disk type * ⓘ

Standard HDD (locally-redundant storage) ▼

Choose Premium SSD disks for lower latency, higher IOPS and bandwidth, and bursting. Single instance virtual machines with Premium SSD disks qualify for the 99.9% connectivity SLA. [Learn more](#)

Delete with VM ⓘ

☒

Key management ⓘ

Platform-managed key ▼

Enable Ultra Disk compatibility ⓘ

☐

Ultra disk is not supported with selected security type.

Data disks

You can add and configure additional data disks for your virtual machine or attach existing disks. This VM also comes with a temporary disk.

LUN	Name	Size (GiB)	Disk type	Host caching	Delete with VM ⓘ
-----	------	------------	-----------	--------------	------------------

Step 4: Setting up network interface and IP address configuration for the VM

Network interface

When creating a virtual machine, a network interface will be created for you.

Virtual network * ⓘ

bcdv-1025-vnet

[Create new](#)

Subnet * ⓘ

default (10.0.0.0/24)

[Manage subnet configuration](#)

Public IP ⓘ

None

[Create new](#)

NIC network security group ⓘ

☐ None

☒ Basic

☐ Advanced

Public inbound ports * ⓘ

☐ None

☒ Allow selected ports

Select inbound ports *

SSH (22)



This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Delete NIC when VM is deleted ⓘ

☐

Step 5: After all validation checks passed deploy the machine and download the pem file

Create a virtual machine ...

✓ Validation passed

Microsoft Defender for Cloud	Basic (free)
System assigned managed identity	Off
Login with Azure AD	Off
Auto-shutdown	Off
Backup	Disabled
Enable hotpatch	Off
Patch orchestration options	Image Default

Monitoring

Alerts	Off
Boot diagnostics	On
Enable OS guest diagnostics	Off

Advanced

Extensions	None
VM applications	None
Cloud init	No
User data	No
Disk controller type	SCSI
Proximity placement group	None
Capacity reservation group	None

Create

< Previous

Next >

VISUAL STUDIO CODE

Here we would connect to the vm we created earlier above

Step 6: this is the pem file downloaded from azure vm platform for the vm created above. It allows us to connect to our vm using ssh

```
chmod 400 <keyname>.pem
```

```
darkn@MSI MINGW64 ~/BCDV-1025-LABS/LAB-6 (master)
$ chmod 400 bcdv_1025.pem
```

Step 7: Connect to vm using ssh and key pem file

```
ssh -i <private key path >.pem vmname@vm_ip_address
```



```
darkn@MSI MINGW64 ~/BCDV-1025-LABS/LAB-6 (master)
$ ssh -i bcdv_1025.pem azureuser@20.104.76.244
```

System Setup: Prerequisites

Update package lists:

```
sudo apt update
```

Upgrade installed packages:

```
sudo apt -y upgrade
```

Install curl:

```
sudo apt install curl
```

Install git:

```
sudo apt install git
```

Install python:

```
sudo apt install python
```

Install Go and jq:

```
Sudo apt-get install jq
```

```
Sudo apt-get install golang
```

Install additional dependencies:

```
sudo apt install apt-transport-https ca-certificates gnupg-agent
software-properties-common
```

Install Docker:

Import Docker's GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
```

Add Docker repository:

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
```

Update package lists:

```
sudo apt update
```

Install Docker:

```
sudo apt -y install docker-ce
```

Add user to the docker group:

```
sudo usermod -aG docker <username>
```

Install Docker Compose:

Download Docker Compose:

```
sudo curl -L  
https://github.com/docker/compose/releases/download/1.27.4/docke  
r-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-  
compose
```

Set executable permissions:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Clone Hyperledger Fabric Samples:

Clone the repository:

```
git clone https://github.com/hyperledger/fabric-samples.git
```

Change to the cloned directory:

```
cd fabric-samples
```

Set up the Test Network:

Run the bootstrap script:

```
curl -sSL  
https://raw.githubusercontent.com/hyperledger/fabric/main/script  
s/bootstrap.sh | bash -s
```

Change to the test network directory:

```
cd fabric-samples/test-network
```

Deploy the Chaincode:

Bring down the network:

```
./network.sh down
```

Bring up the network and create a channel:

```
./network.sh up createChannel -ca -s couchdb
```

Deploy the chaincode:

```
./network.sh deployCC -ccn autorium_monitor_ -ccv 1.0 -ccp  
../../autorium/autorium_monitor -ccl javascript
```

Our Chaincode

The code:

```
"use strict";

const { Contract } = require("fabric-contract-api");

class TerrariumMonitor extends Contract {
  // Initialize the ledger with sample device data
  async initLedger(ctx) {
    console.info("===== START : Initialize Ledger  
=====");
    const devices = [
      {
        doctype: "device",
        deviceId: "device1",
        owner: "Client",
        sensorData: [
          { temperature: 22.5, humidity: 40, timestamp: Date.now() },
          { temperature: 23.1, humidity: 42, timestamp: Date.now() },
        ],
      },
      // Add more devices here
    ];

    // Store the devices in the ledger
    for (let i = 0; i < devices.length; i++) {
      await ctx.stub.putState(
        "DEVICE" + i,
        Buffer.from(JSON.stringify(devices[i]))
      );
    }
  }
}
```

```

    );
    console.info("Added <--> ", devices[i]);
  }
  console.info("===== END : Initialize Ledger =====");
}

// Query a device by its deviceId
async queryDevice(ctx, deviceId) {
  const deviceAsBytes = await ctx.stub.getState(deviceId);
  if (!deviceAsBytes || deviceAsBytes.length === 0) {
    throw new Error(`${deviceId} does not exist`);
  }
  console.log(deviceAsBytes.toString());
  return deviceAsBytes.toString();
}

// Create a new device with the given deviceId and owner
async createDevice(ctx, deviceId, owner) {
  console.info("===== START : Create Device =====");

  const device = {
    doctype: "device",
    deviceId,
    owner,
    sensorData: [],
  };

  await ctx.stub.putState(deviceId,
    Buffer.from(JSON.stringify(device)));
  console.info("===== END : Create Device =====");
}

// Record sensor data for a device

```

```

    async recordSensorData(ctx, deviceId, data) {
        console.info("===== START : Record Sensor Data
=====");

        const deviceAsBytes = await ctx.stub.getState(deviceId);
        if (!deviceAsBytes || deviceAsBytes.length === 0) {
            throw new Error(`${deviceId} does not exist`);
        }
        const device = JSON.parse(deviceAsBytes.toString());
        device.sensorData.push(data);

        await ctx.stub.putState(deviceId,
Buffer.from(JSON.stringify(device)));
        console.info("===== END : Record Sensor Data
=====");
    }

    // Get all devices stored in the ledger
    async getAllDevices(ctx) {
        console.info("===== START : Get All Devices =====");

        const startKey = "";
        const endKey = "";

        const iterator = await ctx.stub.getStateByRange(startKey, endKey);

        const allDevices = [];
        while (true) {
            const result = await iterator.next();

            if (result.value && result.value.value.toString()) {
                console.log(result.value.value.toString());
                allDevices.push(JSON.parse(result.value.value.toString()));
            }
        }
    }

```

```

        if (result.done) {
            await iterator.close();
            console.info("===== END : Get All Devices
=====");
            return JSON.stringify(allDevices);
        }
    }
}
}

module.exports = TerrariumMonitor;

```

The functionality of the code:

The code defines a custom contract named `MyContract`, which extends the `Contract` class from the `fabric-contract-api` package.

The `initLedger` function initializes the ledger with sample device data. It creates and stores device objects with their respective device IDs, owners, and sensor data.

The `queryDevice` function allows querying for a specific device by its device ID. It retrieves the device's data from the ledger and returns it.

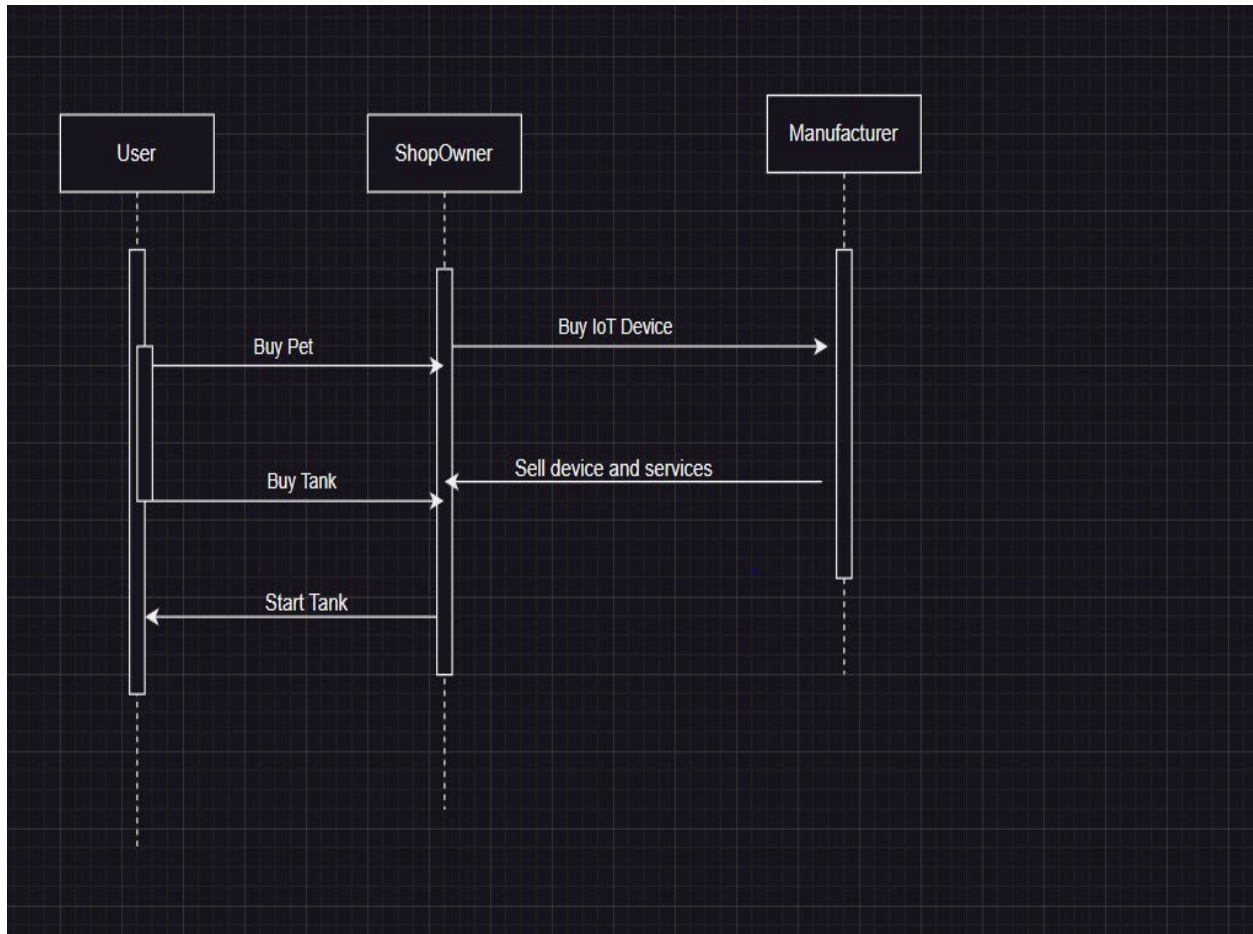
The `createDevice` function creates a new device by providing a device ID and owner. It creates a device object with empty sensor data and stores it in the ledger.

The `recordSensorData` function allows recording sensor data for a specific device. It retrieves the device's data from the ledger, appends the new sensor data to the existing data, and updates the ledger with the modified device object.

The `getAllDevices` function retrieves all devices stored in the ledger. It iterates over the range of keys in the ledger and retrieves each device's data, collecting them into an array. The array of devices is then returned as a JSON string.

These functions collectively enable the management of devices and sensor data in the Terrarium Monitoring system on the Hyperledger Fabric blockchain

LOGICAL ILLUSTRATION OF OUR SMART CONTRACT



References

<https://github.com/hyperledger/fabric-samples>

<https://hyperledger-fabric.readthedocs.io/en/release-2.5/prereqs.html>

<https://hyperledger-fabric.readthedocs.io/en/release-2.5/install.html>

https://hyperledger-fabric.readthedocs.io/en/release-2.5/test_network.html

https://hyperledger-fabric.readthedocs.io/en/release-2.5/test_network.html#bring-up-the-test-network