Blockchain App in a Day Workshop

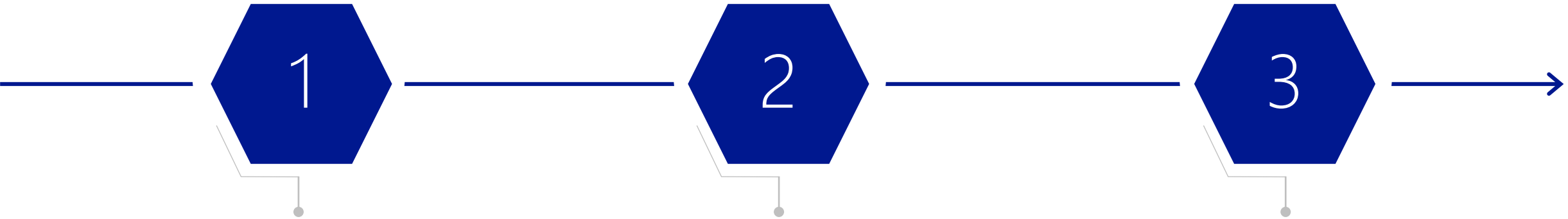
PJ Johnson,
Architect, Microsoft Technology Center

Introduction

- What is Azure building?
- What is Azure Blockchain Workbench?

Building an end-to-end blockchain app is a huge undertaking

So we've taken steps to create a platform that would tackle those challenges

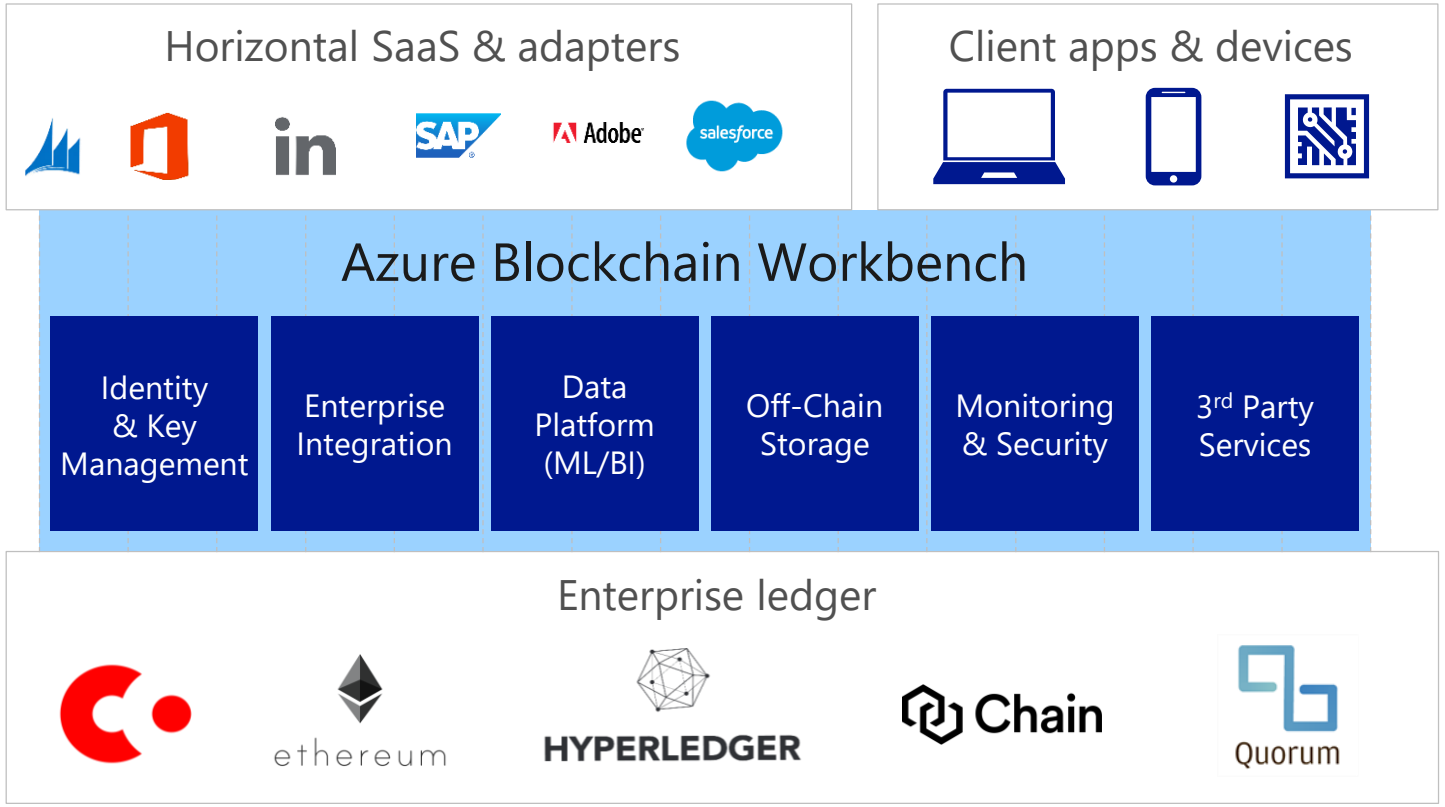


Populated preconfigured networks

Integrated relevant cloud services

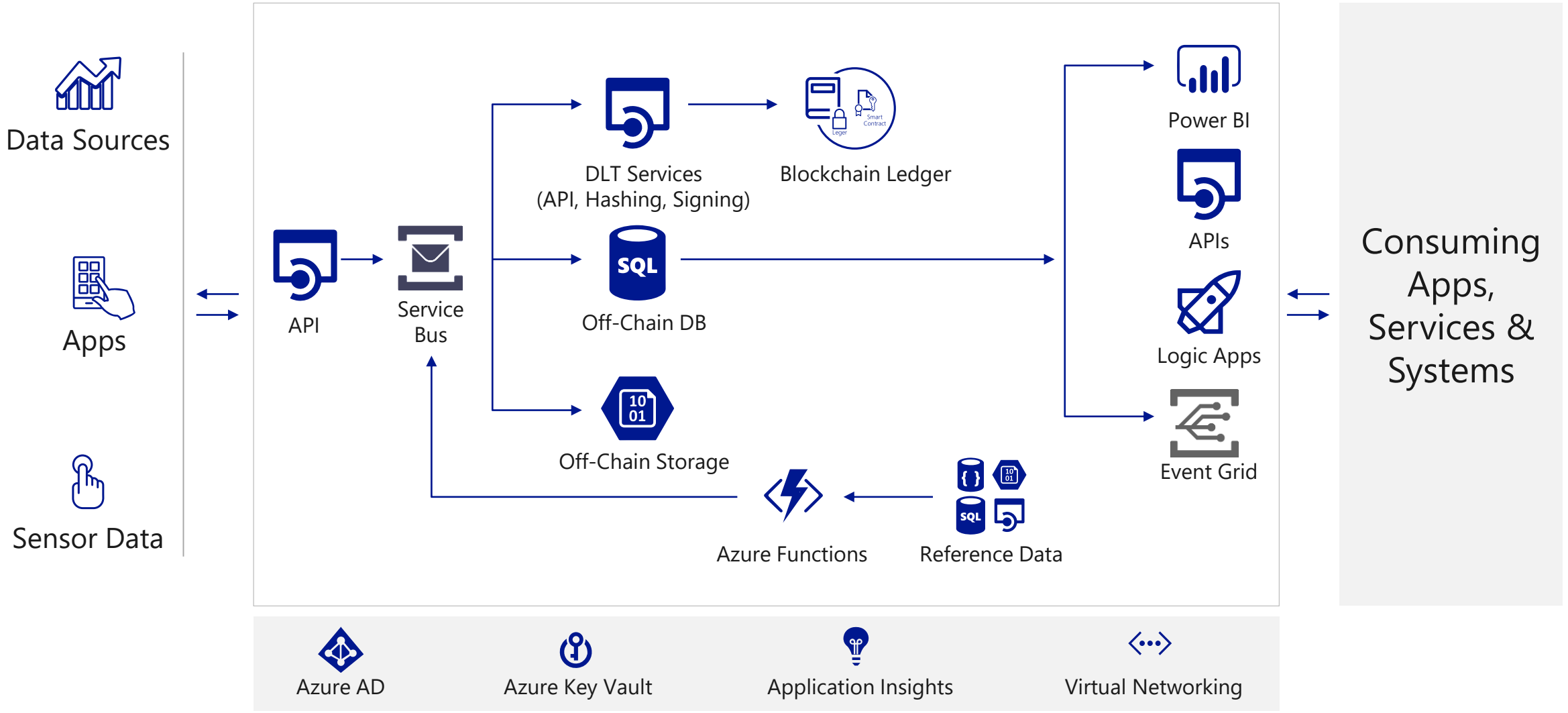
Built a simple interface

With Azure Blockchain Workbench you can skip the scaffolding

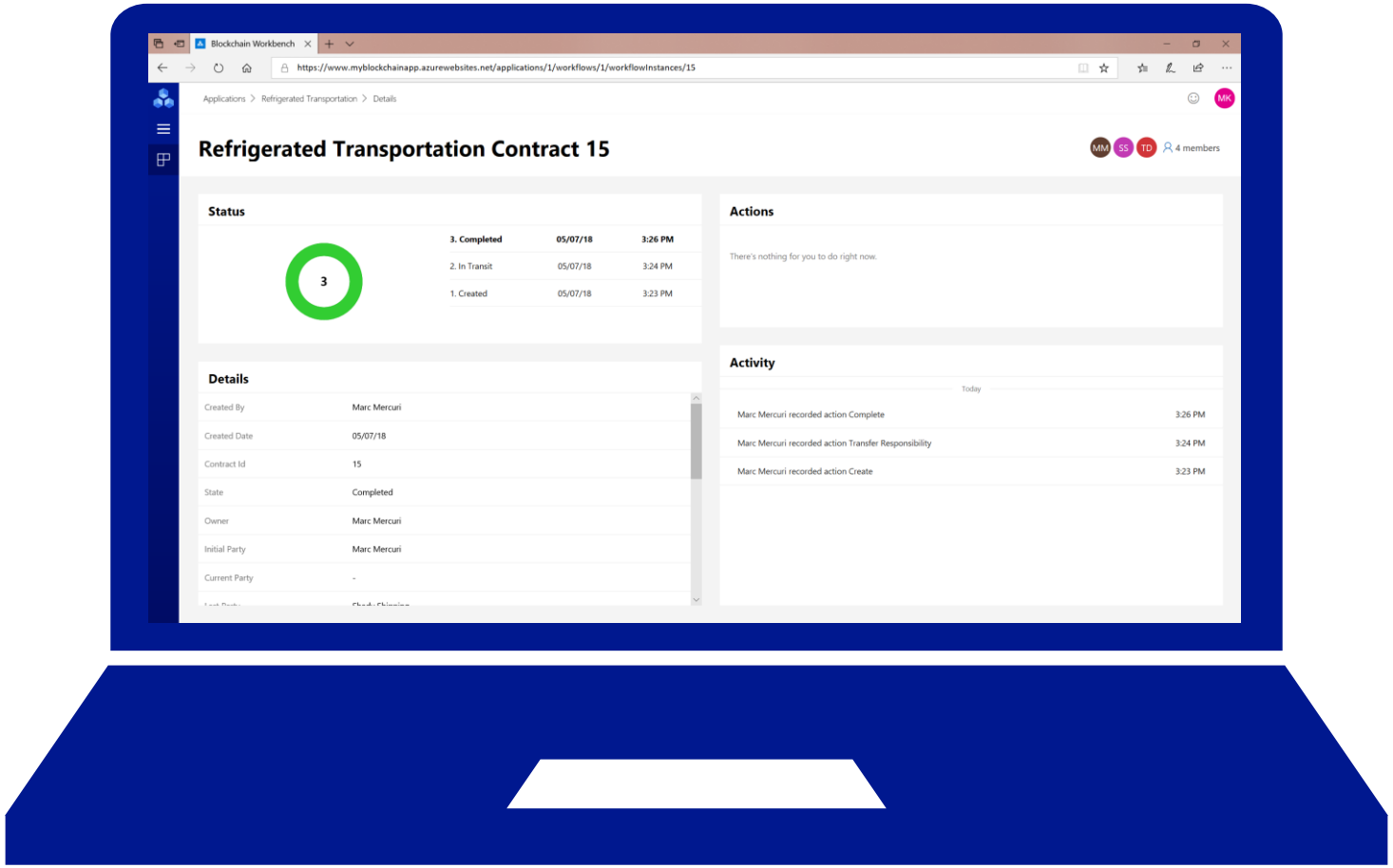


- Workflow execution
- Identity & key management
- Ledger-neutral approach
- Auto-generate starter apps
- Integration APIs & events
- Workflow/user admin
- Azure data integration

With an enterprise-ready, customizable approach



Now, we've built a simple interface for deploying these services and interacting with smart contracts



Workbench lets you focus your time on adding value for your business

Demonstration - Refrigerated Transportation

- App story
- App roles
- App contract states
- State transitions
- Function definitions
- IoT integration – example of extending Azure Blockchain Workbench

Deploy – Asset Transfer

Detailed Review:

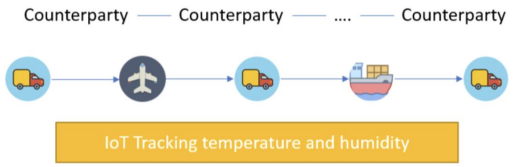
- App story
- App roles
- App contract states
- State transitions
- Solidity code
- JSON configuration

Deploy the Asset Transfer application into your Azure Blockchain Workbench

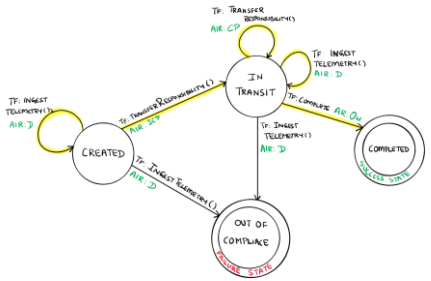
Smart Contract Application Development

- Create application story
- Define application roles, states, workflow, and data elements
- Code Solidity and JSON smart contract application files
- Deploy Solidity and JSON files using Workbench
- Create Azure AD users, add members as application users
- Test application

Blockchain App in a Day - Process Flow



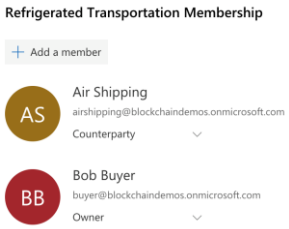
Step 1 | Create Story



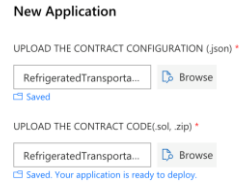
Step 4 | Draft Workflow

Application Role	Available Functions	Parameters
InitiatingCounterParty	Create contract instance Transfer responsibility	MinHumidity MaxHumidity MinTemperature MaxTemperature Device Name Counterparty Name Owner Name
Counterparty	Transfer responsibility	Counterparty Name
Device	Ingest telemetry	ComplianceSensorReading LastSensorUpdateTimestamp
Owner	Mark delivery complete	---
Observer	---	---

Step 5 | Specify On-Chain Data



Step 9 | Add Users and Test



Step 8 | Deploy App

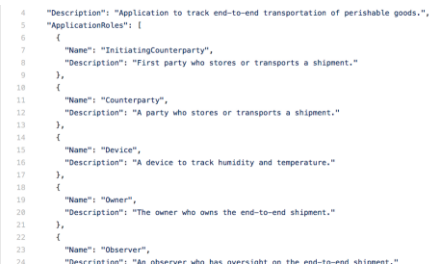
Name	Description
InitiatingCounterParty	The first participant in the supply chain.
Counterparty	A party to whom responsibility for a product has been assigned. For example, a shipper
Device	A device used to monitor the temperature and humidity of the environment the good(s) are being shipped in.
Owner	The organization that owns the product being transported. For example, a manufacturer
Observer	The individual or organization monitoring the supply chain. For example, a government agency

Step 2 | Define App Roles

Name	Description
Created	Indicates that the contract has initiated and tracking is in progress.
InTransit	Indicates that a Counterparty currently is in possession and responsible for goods being transported.
Completed	Indicates the product has reached it's intended destination.
OutOfCompliance	Indicates that the agreed upon terms for temperature and humidity conditions were not met.

Step 3 | Define Smart Contract States

Step 6 | Code Smart Contract



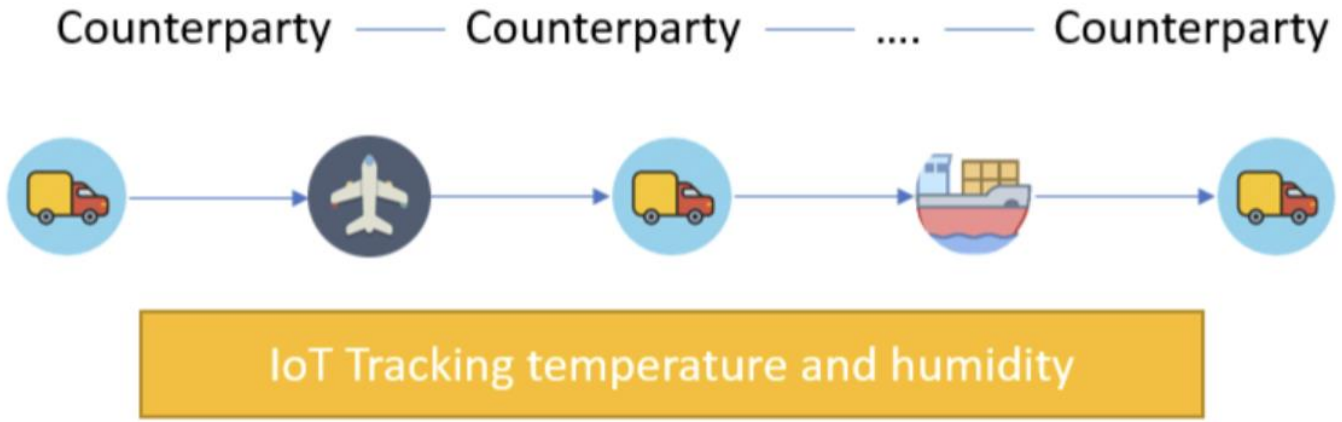
Step 7 | Code Config

Create your application story

Start with a Scenario Overview

The refrigerated transportation smart contract covers a provenance scenario with IoT monitoring. You can think of it as a supply chain transport scenario where certain compliance rules must be met throughout the duration of the transportation process. The initiating counterparty specifies the humidity and temperature range the measurement must fall in to be compliant. At any point, if the device takes a temperature or humidity measurement that is out of range, the contract state will be updated to indicate that it is out of compliance.

Illustrate the desired process flow



Source: *https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation*

Define your application roles

Name	Description
InitiatingCounterParty	The first participant in the supply chain.
Counterparty	A party to whom responsibility for a product has been assigned. For example, a shipper
Device	A device used to monitor the temperature and humidity of the environment the good(s) are being shipped in.
Owner	The organization that owns the product being transported. For example, a manufacturer
Observer	The individual or organization monitoring the supply chain. For example, a government agency

All participants can view the state and details of the contract at any point in time. The counterparty doing the transportation will specify the next counterparty responsible, and the device will ingest temperature and humidity data which gets written to the chain. This allows the Supply Chain Owner and Supply Chain Observer to pinpoint which counterparty did not fulfill the compliance regulations if at any point in the process either the temperature or humidity requirements were not met.

Source: *https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation*

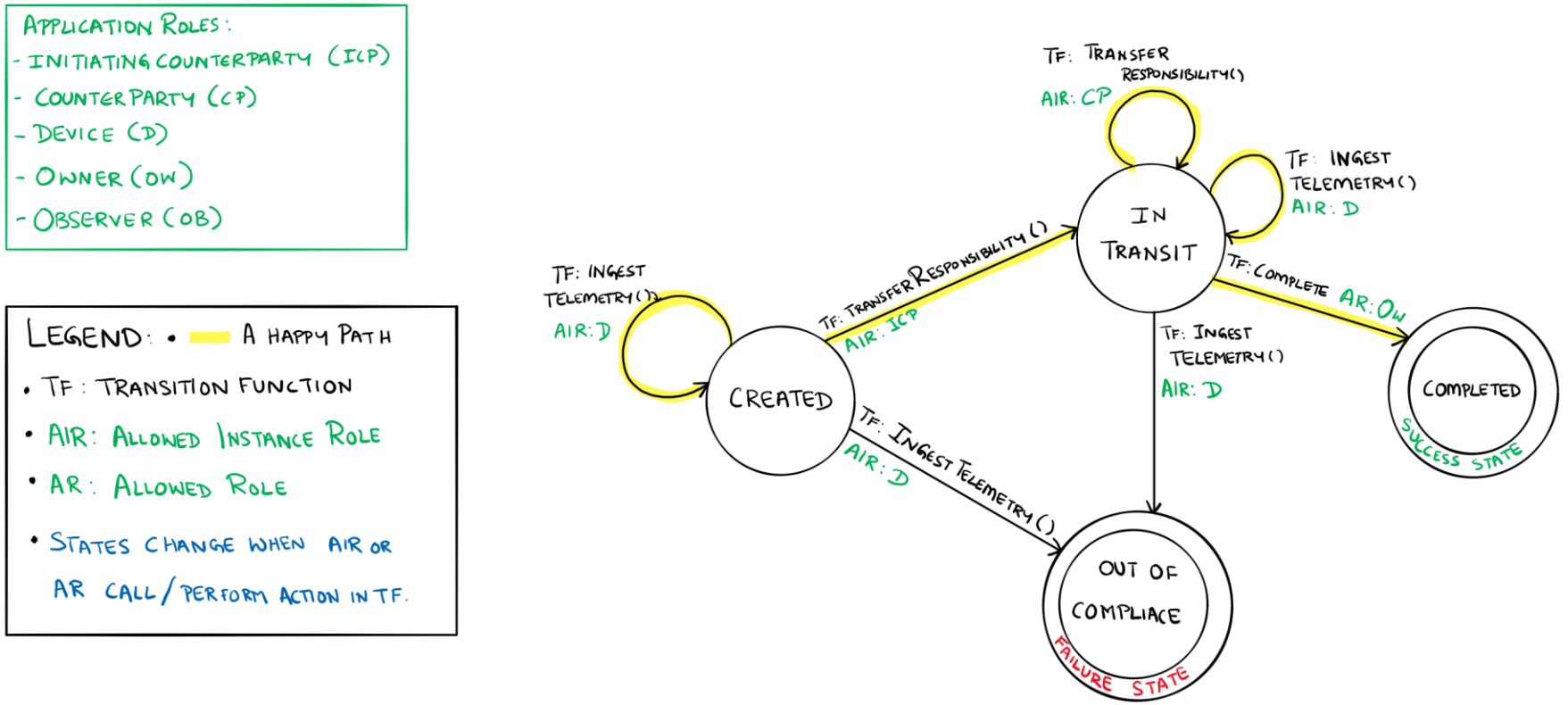
Enumerate all of your smart contract states

Name	Description
Created	Indicates that the contract has initiated and tracking is in progress.
InTransit	Indicates that a Counterparty currently is in possession and responsible for goods being transported.
Completed	Indicates the product has reached it's intended destination.
OutOfCompliance	Indicates that the agreed upon terms for temperature and humidity conditions were not met.

Combine the application story, roles and smart contract states to create a state transition design that articulates the possible flows, and the various transition functions at each state. Each user is only allowed to take certain actions depending on the application role. Instance roles indicate that only the user with the application role assigned to the specific contract is able to take actions on the contract.

Source: *https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation*

Draft the workflow details as a state transition design



This contract demonstrates how to collect telemetry information and enforce contract specifics related to conditions during transport. Specifically, receiving and evaluating temperature and humidity data against an agreed upon acceptable range. If the IoT device identifies that the telemetry is out of the acceptable range, the contract will shift into an out of compliance state and appropriate remedies can be sought. In the highlighted happy path, the device ingests readings, which are in compliance throughout the transportation process, while the involved counterparties transfer responsibility until the transportation is completed.

Source: https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation

Specify data as parameters relative to roles and functions

Application Role	Available Functions	Parameters
InitiatingCounterParty	Create contract instance Transfer responsibility	MinHumidity MaxHumidity MinTemperature MaxTemperature Device Name Counterparty Name Owner Name
Counterparty	Transfer responsibility	Counterparty Name
Device	Ingest telemetry	ComplianceSensorReading LastSensorUpdateTimestamp
Owner	Mark delivery complete	---
Observer	---	---

Use the workflow details to inform your decisions for a minimum viable set of parameters to store on your distributed ledger.

Prepare to write your workflow details as a Solidity Smart Contract

Workbench uses both the configuration file and smart contract code file to create a blockchain application. There is a relationship between what is defined in the configuration and the code in the smart contract.

Base Class

- WorkbenchBase base class enables Blockchain Workbench to create an update the contract
- The base class is required for Blockchain Workbench specific smart contract code
- Your contract needs to inherit from the WorkbenchBase base class

```
pragma solidity ^0.4.20;

contract WorkbenchBase {
    event WorkbenchContractCreated(string applicationName, string workflowName,
    event WorkbenchContractUpdated(string applicationName, string workflowName,

    string internal ApplicationName;
    string internal WorkflowName;

    function WorkbenchBase(string applicationName, string workflowName) interna
        ApplicationName = applicationName;
        WorkflowName = workflowName;
    }
```

Contract

- Contracts need to inherit from the WorkbenchBase base class
- Pass the application name and the workflow name as arguments
- 1:1 relationship between contracts and config files
- Name matching required

```
contract RefrigeratedTransportation is WorkbenchBase('RefrigeratedTransportation',
```

State Variables

- State variables store values of the state for each contract instance
- The state variables in your contract must match the workflow properties defined in the configuration file

```
//Set of States
enum StateType { Created, InTransit, Completed, OutOfCompliance}
enum SensorType { None, Humidity, Temperature }
```

Constructor

- The constructor defines input parameters for a new smart contract instance of a workflow
- The constructor is declared as a function with the same name as the contract
- Required parameters for the constructor are defined as constructor parameters in the configuration file
- The number, order, and type of parameters must match in both files.

```
function RefrigeratedTransportation(address devi
{
    ComplianceStatus = true;
    ComplianceSensorReading = -1;
    InitiatingCounterparty = msg.sender;
    Owner = InitiatingCounterparty;
    Counterparty = InitiatingCounterparty;
    Device = device;
    SupplyChainOwner = supplyChainOwner;
    SupplyChainObserver = supplyChainObserver;
    MinHumidity = minHumidity;
    MaxHumidity = maxHumidity;
```

Functions

- Functions are the executable units of business logic within a contract
- Required parameters for the function are defined as function parameters in the configuration file
- The number, order, and type of parameters must match in both files.

```
function TransferResponsibility(address newCounterparty) public
{
    // keep the state checking, message sender, and device checks separate
    // to not get cloberred by the order of evaluation for logical OR
    if ( State == StateType.Completed )
    {
        revert();
    }

    if ( State == StateType.OutOfCompliance )
    {
        revert();
    }
```

Code the Solidity smart contract

```
pragma solidity ^0.4.20;

contract WorkbenchBase {
    event WorkbenchContractCreated(string applicationName, string workflowName, address originatingAddress);
    event WorkbenchContractUpdated(string applicationName, string workflowName, string action, address originatingAddress);

    string internal ApplicationName;
    string internal WorkflowName;

    function WorkbenchBase(string applicationName, string workflowName) internal {
        ApplicationName = applicationName;
        WorkflowName = workflowName;
    }

    function ContractCreated() internal {
        WorkbenchContractCreated(ApplicationName, WorkflowName, msg.sender);
    }

    function ContractUpdated(string action) internal {
        WorkbenchContractUpdated(ApplicationName, WorkflowName, action, msg.sender);
    }
}

contract RefrigeratedTransportation is WorkbenchBase('RefrigeratedTransportation', 'RefrigeratedTransportation') {

    //Set of States
    enum StateType { Created, InTransit, Completed, OutOfCompliance}
    enum SensorType { None, Humidity, Temperature }

    //List of properties
    StateType public  State;
    address public  Owner;
    address public  InitiatingCounterparty;
    address public  Counterparty;
    address public  PreviousCounterparty;
    address public  Device;
    address public  SupplyChainOwner;
    address public  SupplyChainObserver;
    int public  MinHumidity;
    int public  MaxHumidity;
    int public  MinTemperature;
    int public  MaxTemperature;
    SensorType public  ComplianceSensorType;
    int public  ComplianceSensorReading;
    bool public  ComplianceStatus;
    string public  ComplianceDetail;
    int public  LastSensorUpdateTimestamp;

    function RefrigeratedTransportation(address device, address supplyChainOwner, address supplyChainObserver, int minHumidity,
    {
        ComplianceStatus = true;
        ComplianceSensorReading = -1;
        InitiatingCounterparty = msg.sender;
        Owner = InitiatingCounterparty;
        Counterparty = InitiatingCounterparty;
        Device = device;
        SupplyChainOwner = supplyChainOwner;
        SupplyChainObserver = supplyChainObserver;
        MinHumidity = minHumidity;
        MaxHumidity = maxHumidity;
```

Inherit Workbench Contract Class

Smart contract state enumeration

Transition functions

```
function IngestTelemetry(int humidity, int temperature, int timestamp) public
{
    // Separately check for states and sender
    // to avoid not checking for state when the sender is the device
    // because of the logical OR
    if ( State == StateType.Completed )
    {
        revert();
    }

    if ( State == StateType.OutOfCompliance )
    {
        revert();
    }

    if (Device != msg.sender)
    {
        revert();
    }

    LastSensorUpdateTimestamp = timestamp;

    if (humidity > MaxHumidity || humidity < MinHumidity)
    {
        ComplianceSensorType = SensorType.Humidity;
        ComplianceSensorReading = humidity;
        ComplianceDetail = 'Humidity value out of range.';
        ComplianceStatus = false;
    }
    else if (temperature > MaxTemperature || temperature < MinTemperature)
    {
        ComplianceSensorType = SensorType.Temperature;
        ComplianceSensorReading = temperature;
        ComplianceDetail = 'Temperature value out of range.';
        ComplianceStatus = false;
    }

    if (ComplianceStatus == false)
    {
        State = StateType.OutOfCompliance;
    }

    ContractUpdated('IngestTelemetry');
}

function TransferResponsibility(address newCounterparty) public
{
    // keep the state checking, message sender, and device checks separate
    // to not get cloberred by the order of evaluation for logical OR
    if ( State == StateType.Completed )
    {
        revert();
    }

    if ( State == StateType.OutOfCompliance )
    {
        revert();
    }
```

Source: *https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation*

Prepare your Workbench JSON application configuration file

Create JSON configuration file that will contain the following application settings

Application Roles

- Based on Personas
- Linked to security groups in Azure Active Directory
- Initiators start the workflow process
- Participants modify workflow data
- Observers have read-only access

Workflow

- Collection of states and actions that models application business logic as a state machine
- Each workflow consists of one or more smart contracts, which represent the business logic in code files.

Constructor

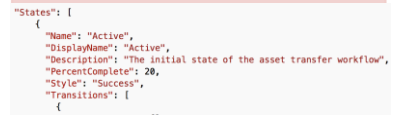
- Defines input parameters for an instance of a workflow
- Maps to the Parameters column of the chart on the previous slide

Functions

- Defines functions that can be executed on the workflow
- The corresponding smart contract must use the same Name for the applicable function
- Maps to the Available Actions on the previous slide

States

- States that define the status within the workflow
- Maps to the current state and target state columns on the previous slide



Transitions

- Available actions to the next state
- One or more user roles may perform an action at each state, where an action may transition a state to another state in the workflow

Code the JSON contract configuration

```
"Description": "Application to track end-to-end transportation of perishable goods.",
"ApplicationRoles": [
  {
    "Name": "InitiatingCounterparty",
    "Description": "First party who stores or transports a shipment."
  },
  {
    "Name": "Counterparty",
    "Description": "A party who stores or transports a shipment."
  },
  {
    "Name": "Device",
    "Description": "A device to track humidity and temperature."
  },
  {
    "Name": "Owner",
    "Description": "The owner who owns the end-to-end shipment."
  },
  {
    "Name": "Observer",
    "Description": "An observer who has oversight on the end-to-end shipment."
  }
],
"Workflows": [
  {
    "Name": "RefrigeratedTransportation",
    "DisplayName": "Refrigerated Transportation",
    "Description": "Main workflow to track end-to-end transportation of perishable goods.",
    "Initiators": [ "Owner" ],
    "StartState": "Created",
    "Properties": [
      {
        "Name": "State",
        "DisplayName": "State",
        "Description": "Holds the state of the contract",
        "Type": {
          "Name": "state"
        }
      },
      {
        "Name": "Owner",
        "DisplayName": "Owner",
        "Description": "The owner of the end-to-end shipment.",
        "Type": {
          "Name": "Owner"
        }
      },
      {
        "Name": "InitiatingCounterparty",
        "DisplayName": "Initial Party",
```

Define application roles

Define workflows

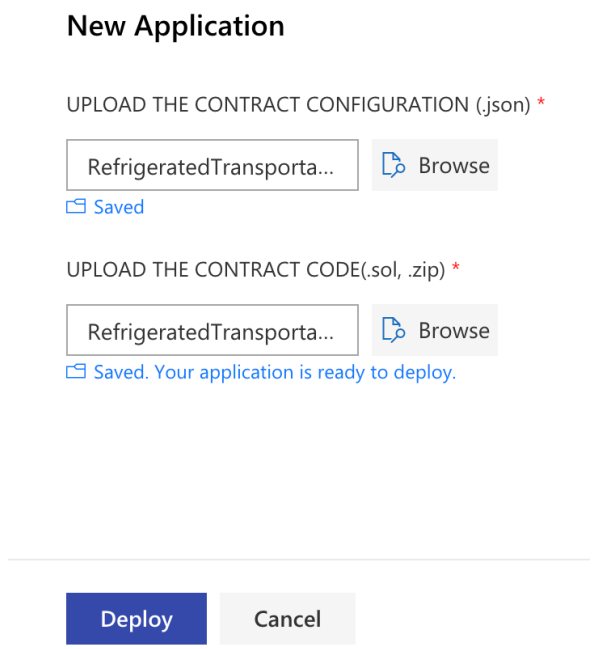
Define parameters used to create smart contract instances and store data

```
      {
        "Name": "device",
        "Description": "...",
        "DisplayName": "Device",
        "Type": {
          "Name": "Device"
        }
      },
      {
        "Name": "supplyChainOwner",
        "Description": "...",
        "DisplayName": "Owner",
        "Type": {
          "Name": "Owner"
        }
      },
      {
        "Name": "supplyChainObserver",
        "Description": "...",
        "DisplayName": "Observer",
        "Type": {
          "Name": "Observer"
        }
      },
      {
        "Name": "minHumidity",
        "Description": "...",
        "DisplayName": "Min Humidity",
        "Type": {
          "Name": "int"
        }
      },
      {
        "Name": "maxHumidity",
        "Description": "...",
        "DisplayName": "Max Humidity",
        "Type": {
          "Name": "int"
        }
      },
      {
        "Name": "minTemperature",
        "Description": "...",
        "DisplayName": "Min Temperature",
        "Type": {
          "Name": "int"
        }
      },
```

Source: *https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation*

Deploy the application

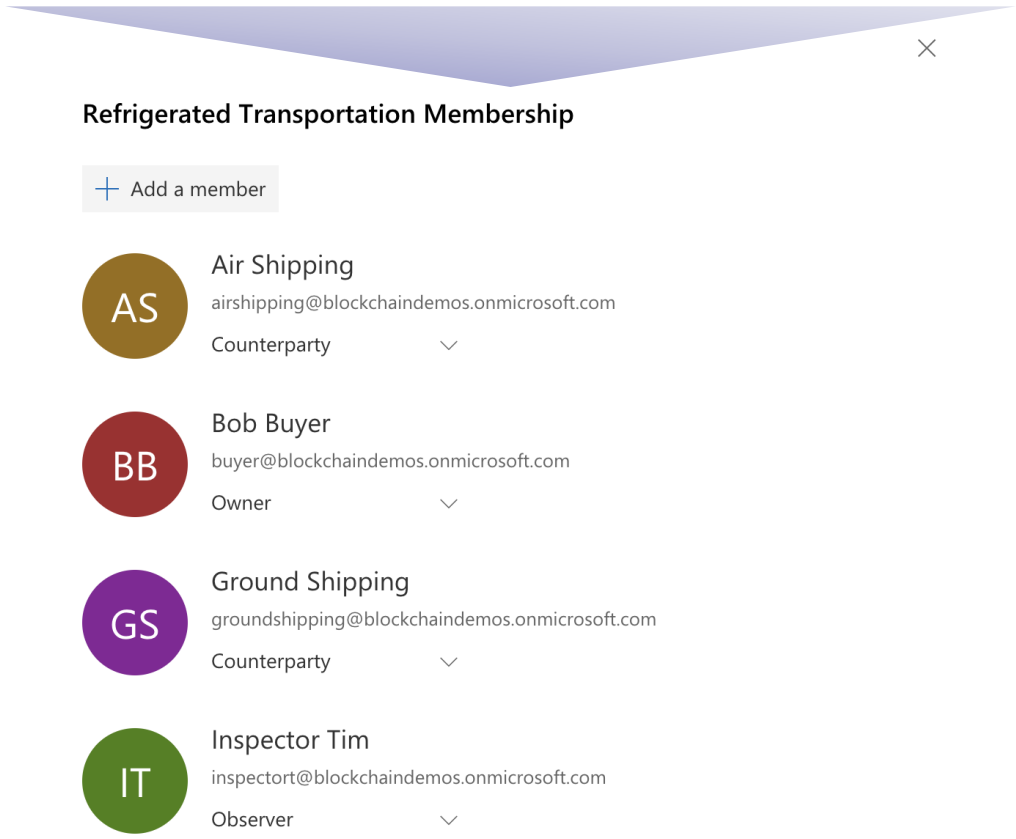
- ☐ **Grant user(s) Administrator application role**
  Refer to link below
- ☐ **Login to Workbench App Service URL**
- ☐ **Click New Application**
- ☐ **Specify path to upload contract configuration json**
- ☐ **Specify path to upload Solidity smart contract file**
- ☐ **Debug if necessary following error messages**
- ☐ **Deploy**



Reference: *https://docs.microsoft.com/en-us/azure/blockchain-workbench/blockchain-workbench-create-app*

Add members to the application

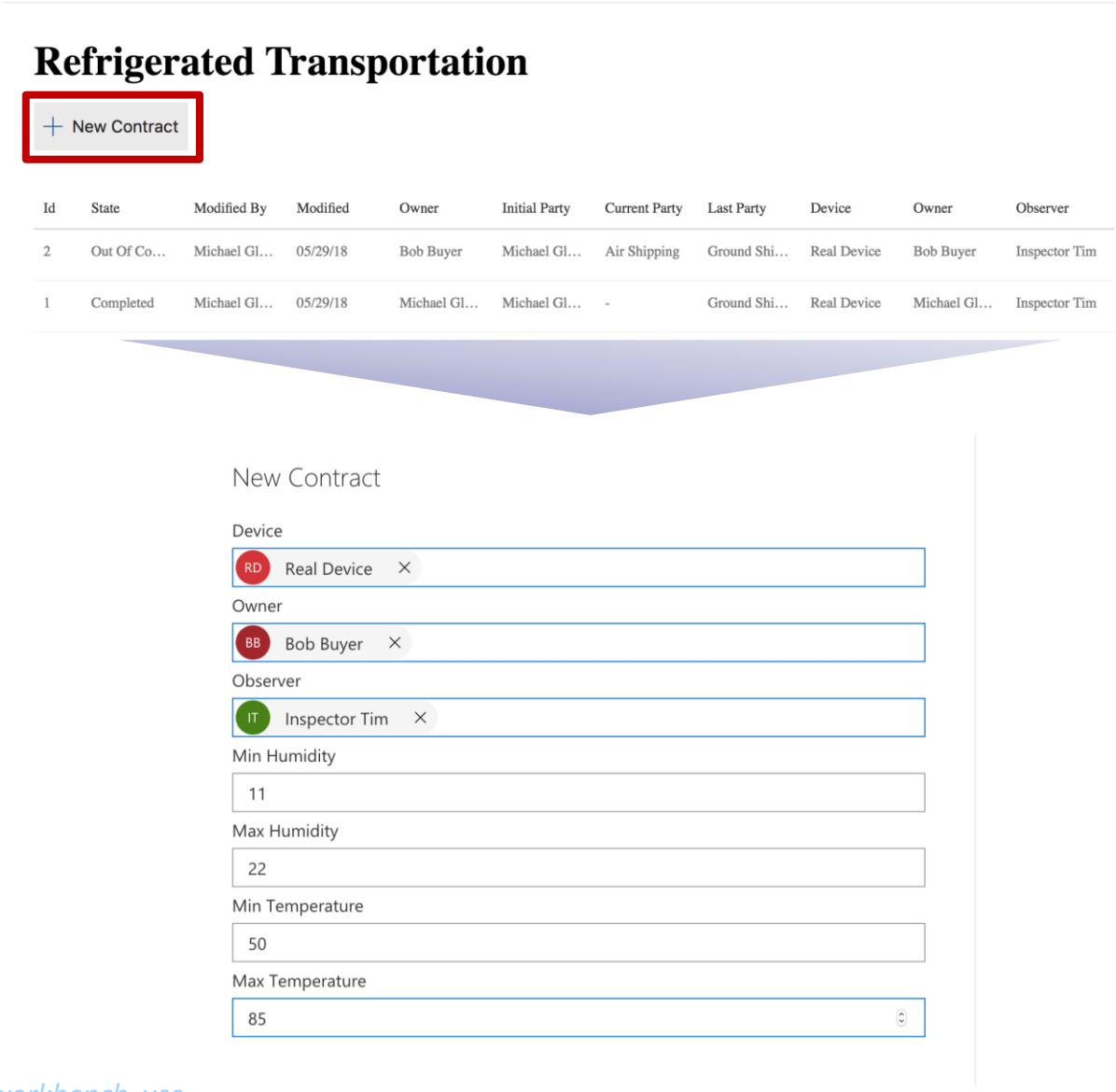
- ☐ **Create new Azure AD users for the roles**
- ☐ **Select by clicking round button in upper right of application tile**
- ☐ **Click Add Member**
- ☐ **Lookup user**
- ☐ **Iterate until you have all the roles necessary for completion of the workflow**



Reference: *https://docs.microsoft.com/en-us/azure/blockchain-workbench/blockchain-workbench-manage-users*

Test the Application

- ☐ **Create a new contract instance**
- ☐ **Specify the initial parameters as defined in the constructor**
- ☐ **Iterate through the test users taking different actions**
- ☐ **Advance to the conclusion of the workflow**



Reference: *https://docs.microsoft.com/en-us/azure/blockchain-workbench/blockchain-workbench-use*