



Robot as a service(RAAS)

By Team Ferrari(Sumeet More and Kapil Deshpande)



Problem statement

Post COVID-19, human interaction will be minimum and most trust will be shown on robotics systems which includes transportation, delivery, manufacturing etc. Right now, we don't have proper system to monetize robotics system efforts. Due to which we cannot measure work done by robotics system and cannot deploy them in different settings. Due to which reusability and asynchronous work is impacting.



What we pitched - I

When it comes to tracking and monetizing assets, blockchain based solutions are heavily effective. Let's break it down in following. Imagine a robot(fork lifter) whose task is to lift heavy load items, generally from docking area or delivery area and delivering it to a plant for production or further processes. Either you can rent such robot or you can buy them. Right now robot's efforts are not considered or valued in manufacturing cost which makes tough to put prize on robot's completed work. Hence we propose a blockchain based system for tracking robot's completed work by assigning wallet to each robot and monetizing task with real world currency.



What we pitched - II

Consider a robot's identity as smart contract and each smart contract holds some token value which can act as wallet for that robot. - We can store state of the robot in that smart contract. eg: we can have a property in smart contract saying `is_robot_available` and if robot is assigned that property will turn true and which will eventually on broader scheme will denote that robot is occupied. - As we know, to change/write to smart contract we require transaction cost and we can put condition on state change which further can be extended as cost for the robot to be assigned for your work. - Each robot if associated with smart contract will come with balance. As robot performs a task, wallet total cost will be reduced and this way one can track expenditure of that robot. - There are can different types of robots and according cost can be accrued. - This way imagining each robot with a smart contract associated to it can help to monetize robot's work and give more insights in terms of costing. - From business view this implementation can be viewed as RAAS(Robots as a service).



Our solution from pitch - I

Soul of our situation was to make system that can track robot's efforts in some form or other.

There are two participants in this solution

- Owner : who owns the bot and manages them.
- Customer/End User : who will use the robots/bots to perform tasks in the factory



Our solution from pitch - II

Owner Flow

- Owner will deploy smart contract with robot's name associated with it.
- Owner will go to portal and enter smart contract address of robot and its name.
- Backend api will store that mapping in the DB with status as available
- This list of robots in DB indicates these robots are in pool to get allocated and registered for the task.



Our Solution from pitch - III

End user/ Customer flow

- End User will open node-red dashboard where he/she can see the available robots which can take task
- Once end user selects robots for the task, REST api will be triggered to reserve selected robot for the mentioned task with customer id.
- This REST api will inturn call smart contract which that bot is associated with(remember in owner's flow we created mapping of smart contract address and robot name, it will be used here)
- Once robot goal is reached the robot will automatically call another smart contract method through REST api and make that robot free and record that transaction in DB through another backend api.



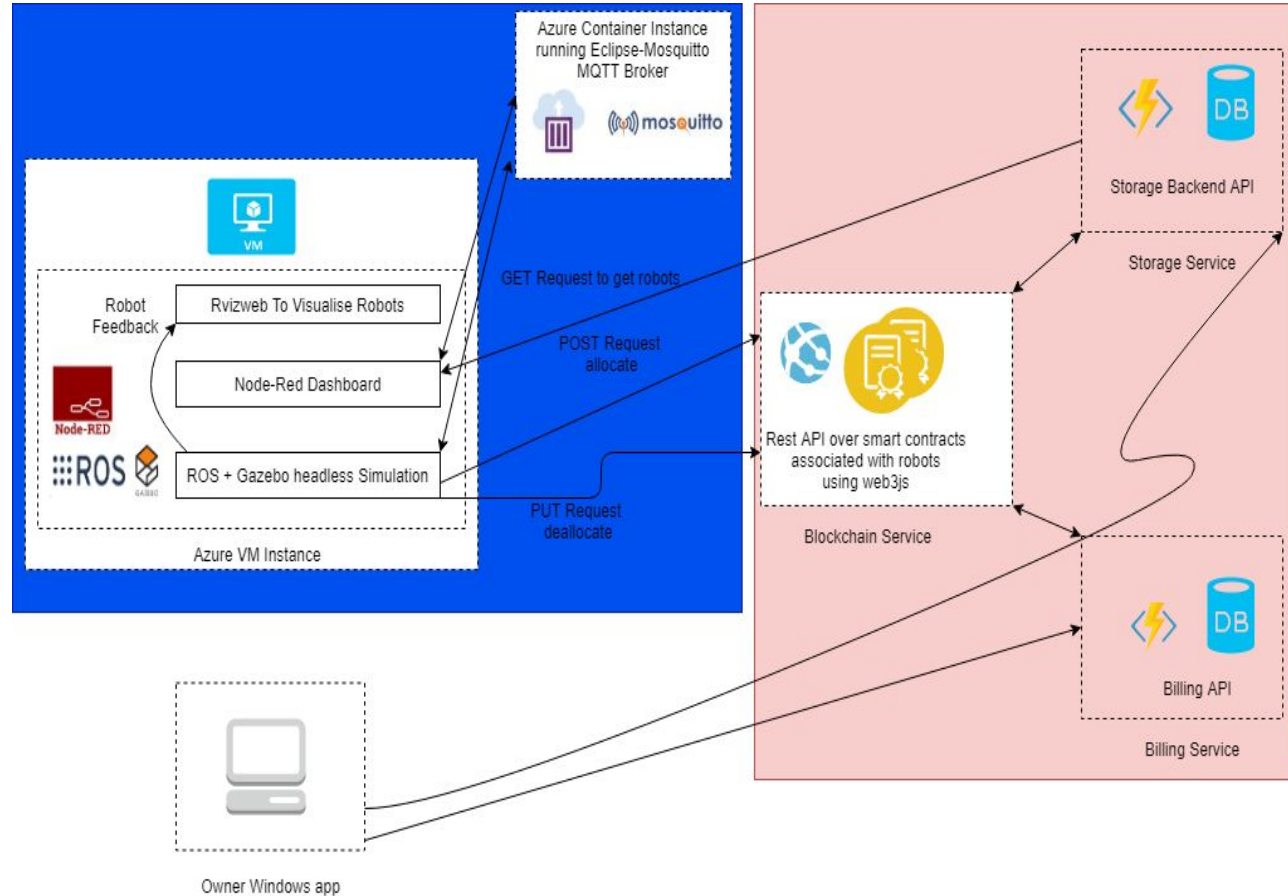
Our Solution from pitch - IV

- Ultimately owner can go to robot web portal and see all the transactions
- He/she can pick each transaction id , go to wallet and check the metadata. This metadata will give information about customerid and total cost of the task.
- Now it is upto the owner whether he/she can give that transaction id to end user and tell them to pay appropriate money as mentioned in transaction or they can generate traditional invoice etc.
- One powerful thing using blockchain helps here is transparency of the transaction and better tracking..



Demo

Architecture





Breakdown - Blockchain - I

```
function Allocate(uint customerid) public returns(string memory response) {  
    if(isActive == false){  
        isActive = true;  
        CustomerId = customerid;  
        return "success";  
    }  
    else{  
        return "failed";  
    }  
}  
  
function Deallocate(uint parameter1,uint standardCost) public returns(string memory response) {  
    if(isActive == true){  
        isActive = false;  
        uint totalCost = parameter1 * standardCost;  
        emit Releasing(CustomerId,totalCost);  
        return "success";  
    }  
    else{  
        return "failed";  
    }  
}
```



Breakdown - Blockchain - II

- Allocate and Deallocate are the core methods of the smart contract. Allocate - reserves bot based on customer name and change its status in smart contract and Deallocate - frees the bot and emit the event which signifies customer id and total cost of the task performed
- Like we said in our pitch/solution, every robot will have its own smart contract so that state of that bot, money in the wallet will be money of bot. **Address of smart contract = address of the robot/identity of the robot.**
- Every bot which is associated with smart contract will have two core methods which we showed in previous slide.
- One challenge we had was to store mapping of smart contract address with robot name. Initially we thought of storing it in the another smart contract but due to memory limits we developed backend api which will store this mapping(remember owner's flow, he adds smart contract address and robot name together - creating that mapping in the backend)



Breakdown - Blockchain - III

- We cannot directly expose smart contracts to robot interface hence we developed rest api as wrapper over it and all communication will happen over it.
- If it is GET request or read only request in smart contract, no private key is required but if one has to change the field/state of the smart contract then private key of the account is required(account which is used while deploying i.e owner's account).
- We can store such private key in azure key vault or config of app service. For prototype phase, we are using it directly in code of rest api but can be moved to any of the services while moving to production
- More about this REST api and backend api in next section of the PPT.



Breakdown - Blockchain - IV

Right now, we are using ethereum on ropsten network but following deployment options can be also used

- Azure blockchain service to deploy in private network(we initially planned to use this service but it is too expensive).
- Ganache in azure VM - we can form 10 blockchain accounts using this service
- Truffle in azure VM - we can form 10 blockchain accounts using this service



Breakdown - APIs - I

We have two APIs - 1) Storage API 2) REST api over smart contract 3) Billing service

Storage API with methods -(we used azure durable entity function so we don't have to use DB explicitly)

- Create Robot - This method will be used to create mapping between smart contract address and robot's name and store in in DB. Owner will use this endpoint through portal when creating bot.
- Allocate Robot - We maintain list of robots but we have to make sure which are robots free & which are not. This api is used when robot is allocated and it makes that particular robot's status unavailable in DB.
- Deallocate Robot - This method works exactly opposite to allocate robot method and makes robot's status available in DB.
- Get robots - Get all available robots only. This api is used by robot interface to showcase end user which robots are available.



Breakdown - APIs - II

Rest API over smart contract has following endpoints/methods. We used Nethereum .NET library to develop rest interface over smart contract methods.(Nethereum is abstraction over web3js library).

- Allocate - This endpoint gets robot name as input. This method first calls backend storage api to get mapping of robot name's and its smart contract address. Then calls smart contract allocate method and once that call is success, it calls backend storage api to make that bot unavailable in DB.
- Deallocate -This method first calls backend storage api to get mapping of robot name's and its smart contract address.Then calls smart contract deallocate method and once that call is success, it calls backend storage api to make that bot available in DB . Ultimately calls billing service api to add transaction id in the DB and owner can use that information for invoicing.



Breakdown - APIs - III

Billing API has following endpoint/Method

- Get Transaction - This api is used in owner's portal web interface. This api gives list of transaction ids which smart contract's deallocate method emitted after successful completion. Transaction Id in turn will give customer id and cost of the task they used bot for.
- Add Transaction - This app adds transaction id in storage. This method is called by deallocate rest api after successful completion of task .



Breakdown - Robotics

Robotics part of the application could be divided into 4 parts:-

- ROS + Gazebo robotics simulation
- Bi-directional communication between robot and controller
- Making a controller using Node-Red
- Visualisation of a running robot on a browser



Breakdown - Robotics - I

ROS + Gazebo robotics simulation-

Robot Operating System is a middleware which allows inter process communication and this enables all the sensors to communicate effectively in a robot. Gazebo is a simulation software which connects to the ros master process and simulates robot with its sensors. We have used the open source package from neo_botix GmbH called neo_simulation (https://github.com/neobotix/neo_simulation) for our robot simulation. This package creates a robot simulation with autonomous navigation enabled using Adaptive Monte Carlo Algorithm(AMCL) for localization and elastic band path planner. The Azure VM used is Ubuntu 16.04 running ROS kinetic, Gazebo 7 which allows neo_simulation package to run.



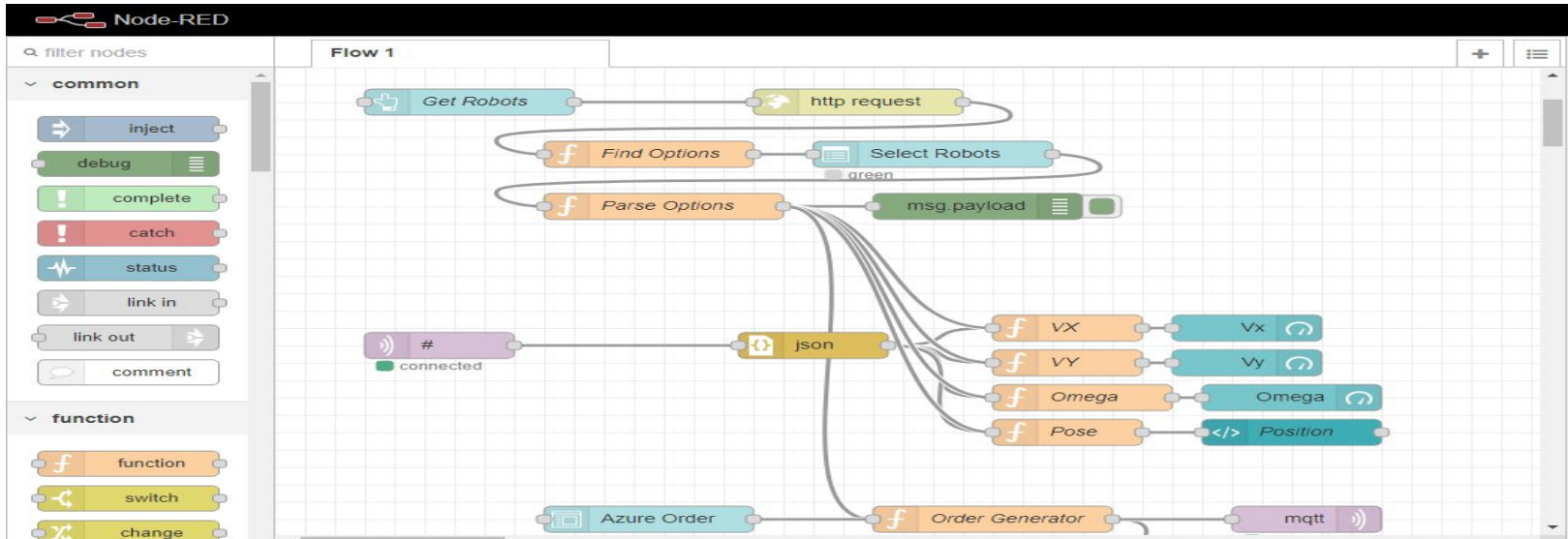
Breakdown - Robotics - II

Bi-directional communication between robot and controller

In any industry whether it is logistics or manufacturing, for controlling robots you need a dedicated master controller panel. So, establishing a bi-directional communication between the robot and the master controller is very important. For the same task, we have used eclipse-mosquitto docker container https://hub.docker.com/_/eclipse-mosquitto running on the azure container instances. This allows controller dashboard(Node-Red Dashboard) to give task to the robot and receive feedback from the robot about velocity and position on topics azure_bot/{name of robot}/move and azure_bot/{name of robot}/fdbck respectively. Since robot name could be changed topic wise the master controller could talk to multiple robots. We tried using Azure IOT Hub but the ROS node for Azure IOT Hub is not stable and does not support all the features mentioned in MQTT v3.1.1 We have used mqtt_bridge package https://github.com/groove-x/mqtt_bridge to convert ROS data to mqtt acceptable json and vice-versa.

Breakdown - Robotics - III

Making a controller using Node-Red




Breakdown - Robotics - III(cont.)

Making a controller using Node-Red


Robot Control Centre

Velocity + Position


Vx



Vy



Omega



X :

0.012

Y :

0.005

theta :

0.274

GET Robots

Select Robots green

GETROBOTS

Azure Order

Azure Order

Customer ID *

x *

y *

theta *

SUBMIT

CANCEL



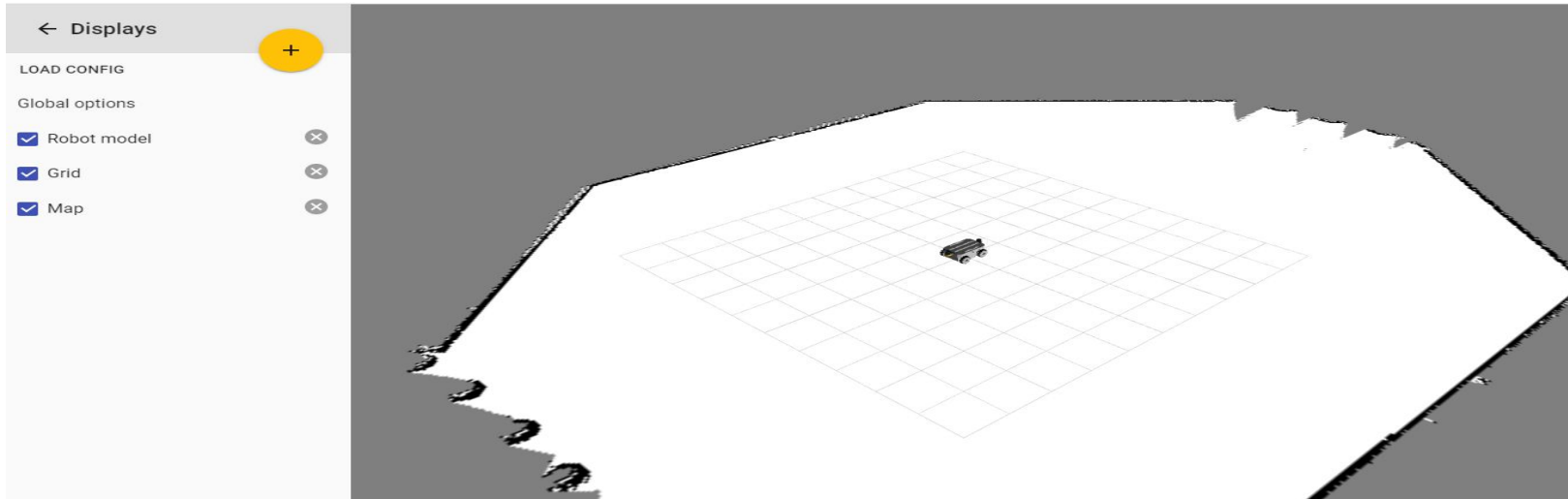
Breakdown - Robotics - III(cont.)

Making a controller using Node-Red

We have installed IBM's Node Red on Azure VM which provides Web browser based editor to connect to IOT devices or connecting to MQTT broker. It also comes with a simple utility to spawn a dashboard which has been used to create master controller screen. This utility is accessible through web over <http://localhost:1880/ui> here localhost is replaced with azure vm public IP. Node-Red is made to run on startup using process manager node module which has been installed on Azure VM as the package 'pm2'.

Breakdown - Robotics - IV

Visualisation of a running robot on a browser





Breakdown - Robotics - IV(cont.)

Visualisation of a running robot on a browser

For Visualisation of the robot over a browser, rvizweb a package provided by osrf (open source robotics foundation) has been used. This package communicates with ros master process and gets two things -

- 1) Robot Model
- 2) Map

These two things are visible on the <http://localhost:8001/rvizweb/www/index.html>, where localhost is replaced with azure vm public IP. In the background this package runs a websocket server to track real position of the robot, collada server to show the robot model.

Our entire solution is on Azure - Here are the services we used

- Azure durable entity function
- ASP.NET core web app hosted on app service.
- Azure VM
- Azure Container Instances





Advantages

- We know immediate question for us is why blockchain? We understand that this could have been implemented in normal system.
 - One of the natural properties of blockchain is that once smart contract is deployed, one cannot manipulate it and its immutability gives confidence to end user that he/she won't be buffed or mischarged for the tasks they chose to perform on robot which is associated to that smart contract.
 - Apart from that, we wanted to leverage transparency of blockchain systems so that whatever pricing is decided, both parties can clearly see and how is computed
 - Only assigning work to robot is done by end user, rest adding value of work and make bot available again will be purely driven by apis and smart contract interactions. This way we eliminate any bias in the system.
- Not everyone can afford buying those expensive industrial robots hence this service can touch medium to large scale of businesses.



Thank You.