ROS-React Web App

Rebecca Carbone, Jefferson Perez Diaz, Mrunal Prakash Gavali, Tamara Abu-Shaar, Mayur Rahangdale, Ricardo Lopez

Agenda

01 **Introduction**

02 **Demo**

03 **Approach & Design**

04 **Conclusion**

Introduction

Framework/Library: Roslib

Technologies used: Roslib,Nav2D, Ros2D, Three, React, Robot Web Tools, Javascript, CSS, AWS, Ngrok

Approach

Why use Roslibjs?

- Able to work with a robot in the browser
- Helpful documentation
- Constant updates
- Lots of libraries available to the public

React

- Most familiar to the group
- User friendly
- Lots of useful functions and libraries

Demo

Robot Control Page

Robot Connected

Position

x: -1.540

y: 1.769

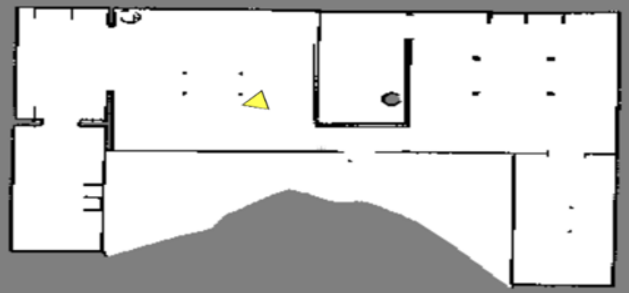
Orientation: 80.58

Velocities

Linear Velocity: -0.000

Angular Velocity: -0.000

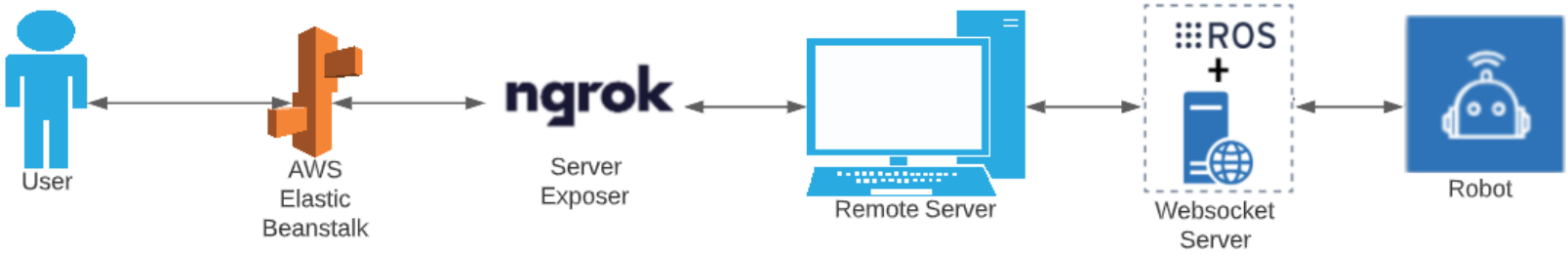
MAP



Dealing with websockets

- Roslibjs requires the use of websockets to connect to its services. To allow any client on the internet to access to those services port forwarding was implemented.
    - What is Port Forwarding?
        - Simply put, it deflects the communication requests from one address to another.
        - Port Forwarding allows any client over the internet to connect to a specific computer or service within a private network.
    - Ngrok
        - Port forwarding can get a little complicated, and it may take time to properly adjust both the environment and the local machine. Ngrok provides a quick and efficient solution to any networked service to the internet without having to set up complicated port forwarding rules.

HIGH LEVEL DESIGN



Libraries and Hello World

- Roslibjs uses websockets to connect with rosbridge and provides publishing, subscribing, and service calls to update the website.
- Rosbridge_server provides a WebSocket connection so browsers can “talk rosbridge”. Roslibjs is a Javascript library for the browser that can talk to ROS via rosbridge_server.
- We retrieved odometry data to update the robots position on the map.
- Used Ros2d library to help post a 2d map of where the robot is currently moving.
- In addition to Ros2d, another library was used (NAV2D) to allow user to navigate through the map by clicking on it.
- Hello World/Get Started: http://wiki.ros.org/roslibjs/Tutorials/BasicRosFunctionality

Set-up and installation

We used **Virtual box** for our **Ubuntu 20.04** linux virtual machine setup to run **ROS Noetic** version of the ROS ecosystem.

1. http://wiki.ros.org/rosbridge_suite --> install rosbridge using following command:

   ```
   Sudo apt-get install ros-noetic-rosbridge-server
   ```

1. Install Roslib in VS code terminal → npm install roslib
2. Install Three → npm install three

Note: All the libraries are added to the project by creating a "js" folder inside "public" folder add libraries content from their respective github repositories inside build folder or CDN min and the from http://robotwebtools.org/ for the following libraries:

1. Roslib.js
2. Ros2d.js
3. Eventemiter2.min.js
4. Easeljs.js
5. Nav2D.js
6. Ros-nav2d-js folder > navigator folder >

   6.1 ImageMapClientNav.js

   6.2 Navigator.js

   6.3 OccupancyGridClient.js

ROS ecosystem: Glossary Terms

1. ***Nodes*** — A node is an executable that uses ROS to communicate with other nodes. A ROS Node can be a *Publisher* or a *Subscriber*.

2. ***Topic*** — A named bus, a pipe between nodes through which the messages flow. It is a collection of messages.

3. ***Messages*** — The predefined format of information that is exchanged between the nodes.

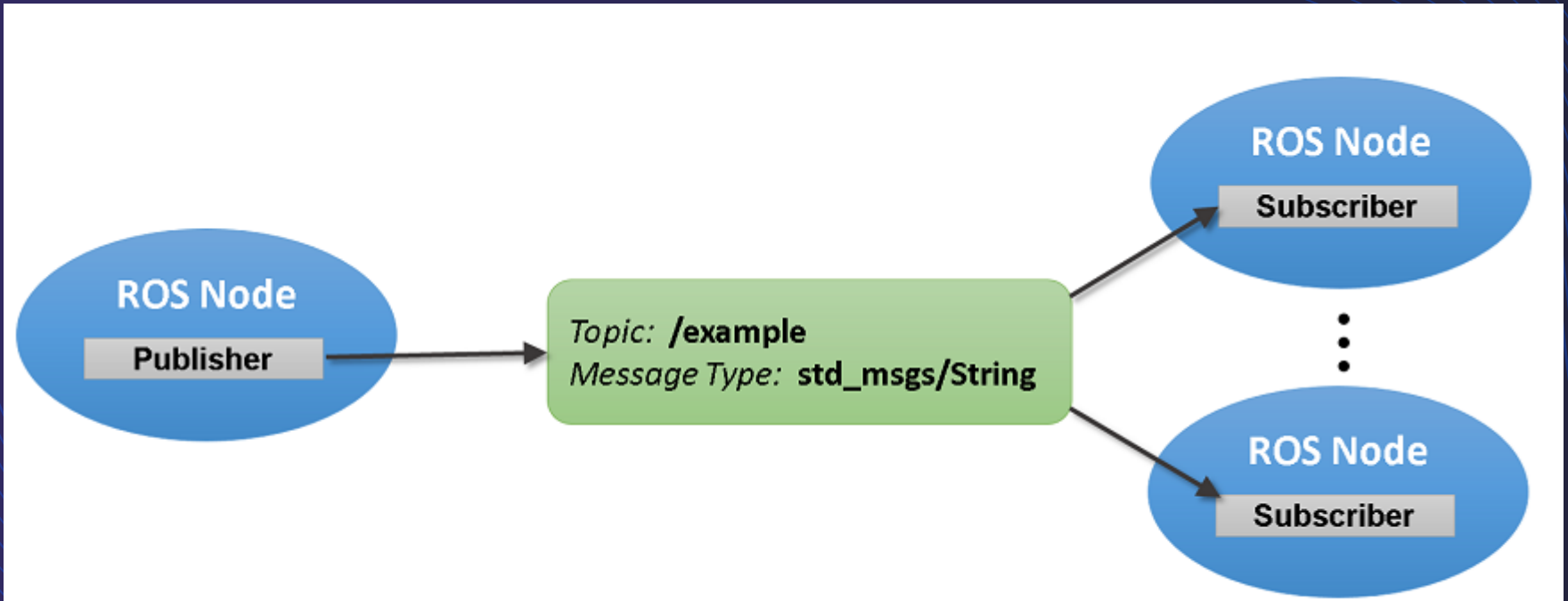
4. **Publisher** — A Publisher is the one puts the messages of some standard *Message Type* to a particular *Topic*. Note that a publisher can publish to one or more *Topic.*

5. **Subscriber** — The Subscriber on the other hand subscribes to the Topic so that it receives the messages whenever any message is published to the Topic. Note that a Subscriber can subscribe to one or more *Topic*.

6. **ROS Master** — Also, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption and all the IP addresses of various nodes are tracked by the ROS *Master*.

Note: ROS master is only part of ROS1 architecture not ROS2.

ROS ecosystem : Publisher subscriber interface



1.1 ROS Publisher Subscriber Model (Courtesy: Mathworks.com)

Joystick - Teleoperation

1. Joystick is programmed using the "react-joystick-component".
   Installed using npm : $ npm install react-joystick-component
1. In this project we developed the application logic for the joystick so that when we move the joystick forward (x-direction) we will send linear velocity commands.
   And when we move the joystick to right (z-direction) we will send "angular velocity commands".
1. The two handles which come with the react-joystick-component are handleMove() and handleStop().

handleMove() code structure:

1. We need to create a ROS publisher on the topic cmd_vel
2. We need to create a twist message to be published to rosbridge
3. We need to publish the message on the cmd_vel topic.

From the documentation of React-joystick component, it is clear that whenever you actually play with the throttle, you're going to raise an event. And this event will tell you how much you have moved in every direction.

To capture the event about how much you have moved in the horizontal direction and the vertical direction, we use the following code: **x: event.y/60** and **z: -event.x/60** for horizontal and vertical direction respectively.

Note: Divided by 60 to make the robot simulation move smoothly.

Turtlebot Simulator setup

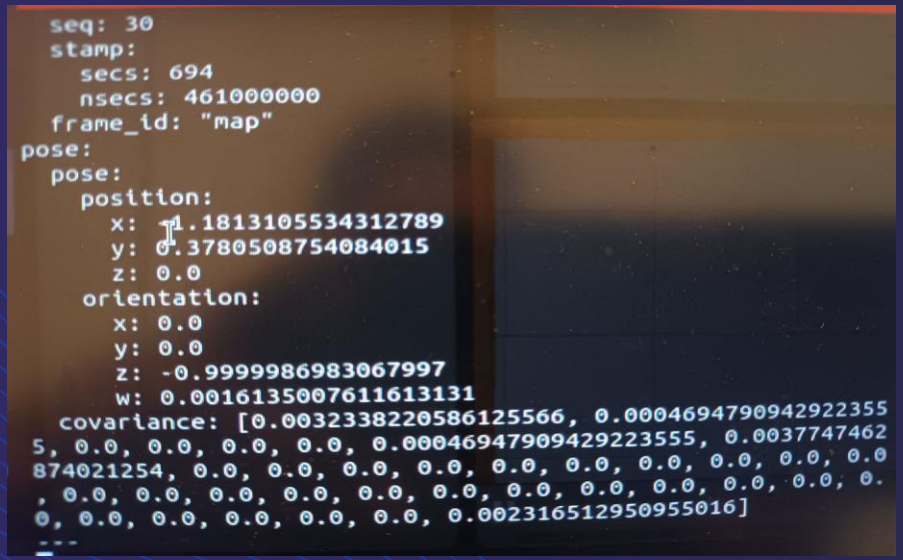
$ sudo apt-get install ros-noetic-joy ros-noetic-teleop-twist-joy \ ros-noetic-teleop-twist-keyboard ros-noetic-laser-proc \ ros-noetic-rgbd-launch ros-noetic-rosserial-arduino \ ros-noetic-rosserial-python ros-noetic-rosserial-client \ ros-noetic-rosserial-msgs ros-noetic-amcl ros-noetic-map-server \ ros-noetic-move-base ros-noetic-urdf ros-noetic-xacro \ ros-noetic-compressed-image-transport ros-noetic-rqt* ros-noetic-rviz \ ros-noetic-gmapping ros-noetic-navigation ros-noetic-interactive-marker

$ sudo apt install ros-noetic-dynamixel-sdk
$ sudo apt install ros-noetic-turtlebot3-msgs
$ sudo apt install ros-noetic-turtlebot3
$ sudo apt install ros-noetic-dynamixel-sdk
$ sudo apt install ros-noetic-turtlebot3-msgs
$ sudo apt install ros-noetic-turtlebot3
$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations
$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
$ cd ~/catkin_ws && catkin_make

$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
$ source /home/j/catkin_ws/devel/setup.bash
$ export TURTLEBOT3_MODEL=burger

Display Robot Position and Orientation

- The RobotState.jsx component code describes the implementation of displaying the robot position, orientation and speed on the web application.
- Rostopic echo amcl_pose will output the position and the orientation of the robot in the terminal in side the virtual machine as shown below:



- The objective is to show this output of the amcl _pose topic on the web application.

Robot Orientation display using Three

- Note that the orientation of the amcl_pose shown in the terminal in the previous slide is in **quaternion** and we converted it to **radians** in the RobotState.jsx.

```
getOrientationFromQuaternion(ros_orientation_quaternion){
    var quat = new Three.Quaternion(
        ros_orientation_quaternion.x,
        ros_orientation_quaternion.y,
        ros_orientation_quaternion.z,
        ros_orientation_quaternion.w
        );
    var RobotOrientation = new Three.Euler().setFromQuaternion(quat);

    return RobotOrientation["_z"] * (180/Math.PI);
}
```

Convert this quaternion into roll, pitch and yaw.

Convert this quaternion into radian.

Display Robot Velocities : Linear & Angular

- Rostopic echo odom will output the velocities of the robot in the terminal in side the virtual machine as shown below:

```
      y: -0.0010588716531599485
      z: -0.7454530561775601
      w: -0.6665562288344046
  covariance: [1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-05,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001]
twist:
  twist:
    linear:
      x: -0.00012054454498057978
      y: -2.838650317321226e-07
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: -0.00012841677241285317
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
---
```

- The objective is to show this output of the odom topic on the web application.

```
//creates a pose callback
velocity_subscriber.subscribe((message) => {
    this.setState({linear_velocity: message.twist.twist.linear.x.toFixed(3)});
    this.setState({angular_velocity: message.twist.twist.angular.z.toFixed(3)});
});
```

Map and Navigation

- To visualize the map as an image, ROS2D was used to publish the map on the web browser so users can interact with it as well as seeing the robot position through two ros topics /map and /robot_pose respectively.

- However, an issue arise when trying to publish the robot's position in the map as we have previously use it to get the state of the robot to get the x and y coordinates.
    - A workable solution was to create an overlay with a different name that would allow us to get the robot's position in the map.

- To navigate the map, using NAV2D enables users to click to any arbitrary position on the map to create a goal for the robot to move to that position. As soon as a goal is given, a marker is created which will be removed from the map once the robot reaches its target.
    - Users can also give the robot an orientation of where they want the robot to be facing, at the moment it gets to the target location by holding the left click.

Running the program locally inside the Linux virtual machine

[terminal 1]

```
$ Roscore
```

[terminal 2]

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

[terminal 3]

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch
map_file:=/home/user/maps/tb1_house.yaml
```

[terminal 4]

```
$ rosrun custom_robot_pose_publisher custom_robot_pose_publisher
```

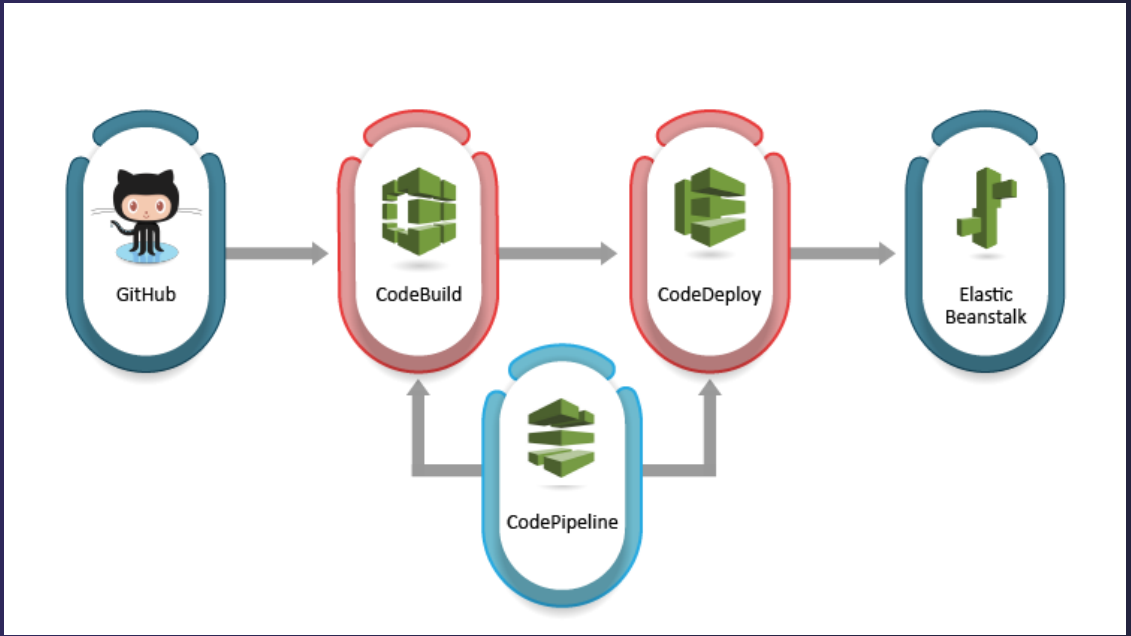
[terminal 5]

```
$ roslaunch rosbridge_server rosbridge_websocket.launch _port:=9090 websocket_external_port:=customPort --
screen
```

Tools used for Continuous Integration and Deployment

CI/CD Stack

- Github and Git bash
- AWS Codepipeline
- AWS Beanstalk



Tools used for Continuous Integration and Deployment

AWS Codepipeline

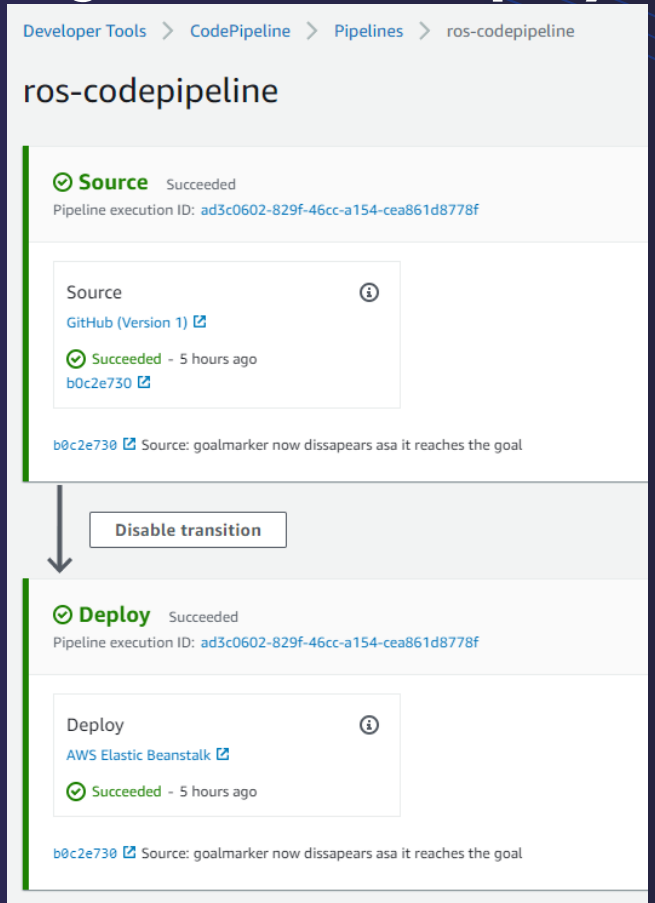
For continuous delivery of web application version from Github.

- Source : github repository comp484final-roswebapp
- Deployed ros-codepipeline pipeline to a new AWS beanstalk instance.

Tools used for Continuous Integration and Deployment

AWS Codepipeline

Source to build stage and deploy stage to AWS beanstalk instance

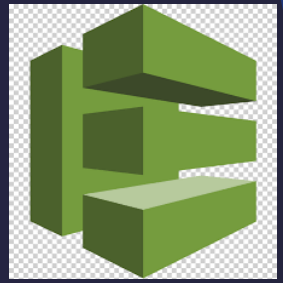


Tools used for Continuous Integration and Deployment

AWS Beanstalk

For continuous deployment of the web application from AWS codepipeline.

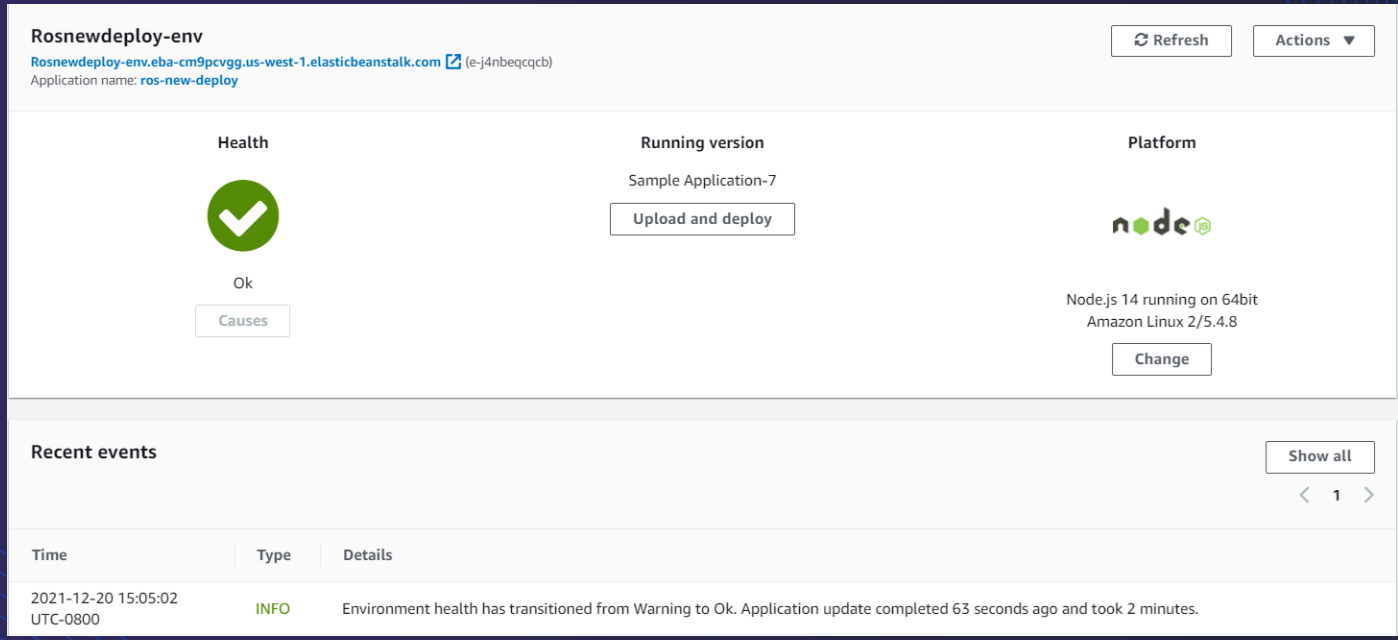
- Beanstalk application : ros-new-deploy
- Beanstalk environment Rosnewdeploy-env
- Env Config: Node.js 14 (64bit Amazon Linux 2/5.4.8)
- Instance health monitoring



Tools used for Continuous Integration and Deployment

AWS Beanstalk

Snapshot of AWS beanstalk instance when updating configuration.



Tools used for Continuous Integration and Deployment

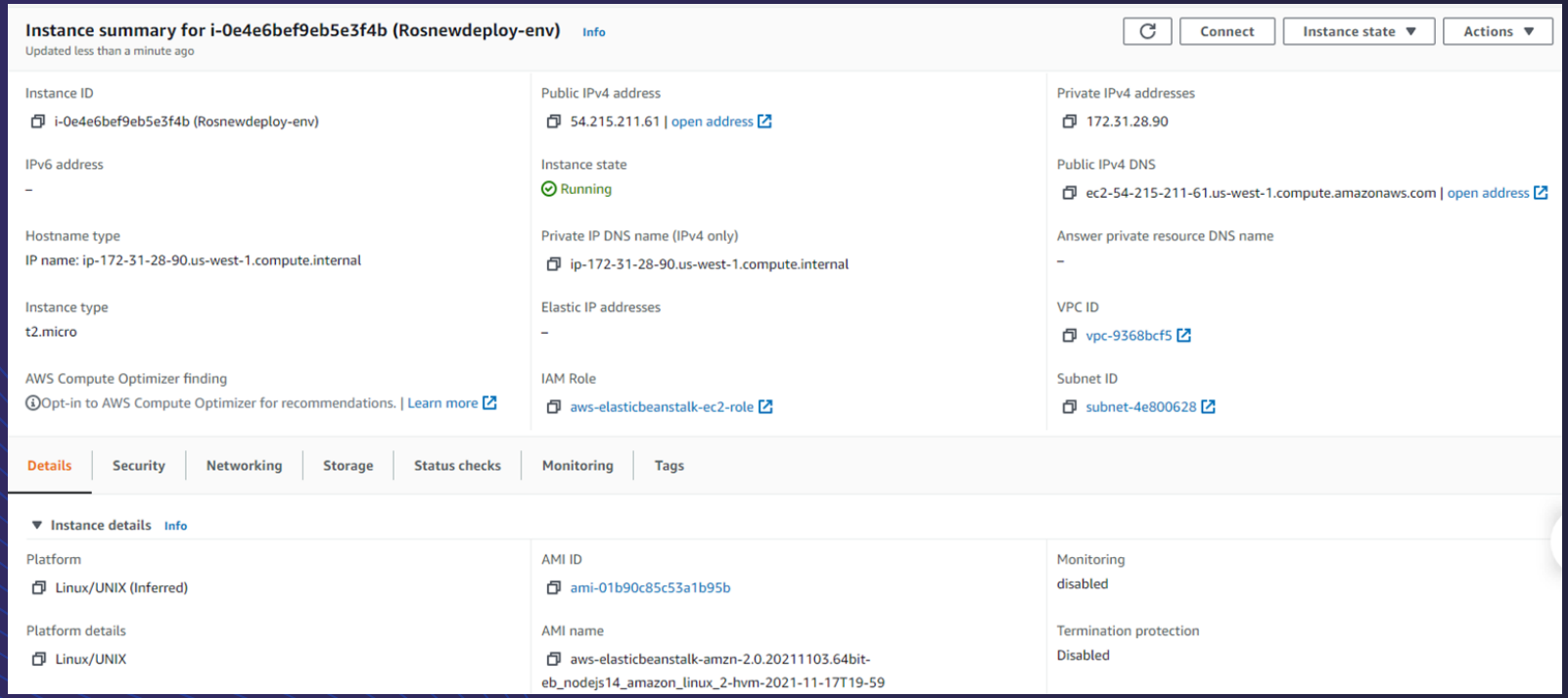
AWS EC2

Monitoring configuration to obtain instance public IP and port.

- Port forwarding ROS server → beanstalk instance
- Public IP 54.215.211.61 and port 8080
- Websocket connections

Tools used for Continuous Integration and Deployment

AWS EC2



Tools used for Continuous Integration and Deployment

Git bash console

To push code changes from local to remote

- 2 git branches for version control
- main branch : configuration
- dev/css branch : UI

Conclusion

Alternative Frameworks (for React):

Angular

- Pros:
  - Doesn't need outside libraries
  - UI configuration is easier/faster
- Cons:
  - Harder to learn

Vue

- Pros:
  - Uses HTML *and* JSX
  - More beginner friendly
- Cons:
  - Not as familiar to our group

Conclusion

Lessons Learned:

- How to work as a team virtually
- Delegate work based on individual ability
- How to work with websockets

Thank You

Check out our github repo at :

https://gitlab.com/mrunalgavali.999/ReactJS_ros_robot_app/-/tree/main/ROS_robot_reactjs_webapp

Check out our website at:

http://rosnewdeploy-env.eba-cm9pcvgg.us-west-1.elasticbeanstalk.com/

References

- robot_state_publisher/Tutorials/Using the robot state publisher on your own robot - ROS Wiki
- http://wiki.ros.org/nav2djs
- http://wiki.ros.org/roslibjs
- http://wiki.ros.org/ros2djs
- https://answers.ros.org/question/355349/rosbridge-websocket-server-over-internet/
- https://ngrok.com/download
- https://emanual.robotis.com/docs/en/platform/turtlebot3/bringup/
- https://alan.app/docs/client-api/web/vanilla/#
- http://wiki.ros.org/rostopic
- https://react-bootstrap.github.io/layout/grid/
- https://stackoverflow.com/questions/53758946/spacing-and-margin-utility-in-react-bootstrap
- https://www.npmjs.com/package/three
- https://medium.com/analytics-vidhya/basics-of-robotics-in-ros-8c9a56d24c6
- https://www.youtube.com/watch?v=MLqH8axh7ss&list=PLWIJHK6SiyBsZ8lnzYWvzyHM5YFssdsZg&index=2
- http://robotwebtools.org/