

Instruction

Introduction

Cloud robotics augments robots by cloud services. Although it is regarded as a revolutionary paradigm, great efforts have to be made to develop cloud services dedicated to robotic scenarios. Instead of developing those services from scratch, we can find that many robotic algorithms have been encapsulated into software packages of special forms (e.g. ROS packages) in the open source community. Although they are not designed for cloud environment originally, could they be rapidly encapsulated and deployed as cloud services? This project aims to address this challenge by developing a Platform-As-A-Service framework which can directly deploy those software packages. The packages can be transformed into Internet-accessible cloud services automatically. Another requirement is that multiple robots should be able to access such a cloud service simultaneously even if the corresponding package is originally designed just to serve a single robot, with the guarantee of Quality of Service for each robot. Docker or other open-source application container technology can be adopted to realize this goal at the backend.

micROS-cloud server

Environment

- Ubuntu 14.04.3 LTS
- Python 2.7.6
- Docker version 1.11.2

Installation

Docker installation

- (1) Follow the tutorial to get Docker installed:
<https://docs.docker.com/engine/installation/linux/ubuntu/linux/>
- (2) Install and Create a Docker Swarm: <https://docs.docker.com/swarm/install-w-machine/>

Flask installation

- (1) Log into your machine as a user with sudo or root privileges.
- (2) Open a terminal window.

- (3) Update package information
- (4) Install python-pip:

```
$ sudo apt-get update
```

```
$ sudo apt-get install python-pip
```
- (5) Install flask:

```
$ sudo pip install flask
```
- (6) Install database:

```
$ sudo pip install flask_sqlalchemy
```
- (7) Install flask_login

```
$ sudo pip install flask_login
```
- (8) Install flask-wtf

```
$ sudo pip install flask-wtf
```

Docker-py installation

```
$ sudo pip install docker-py
```

micROS-cloud server start

- (1) Download and unzip the micROS-cloud package.
Or git clone the package:

```
$ git clone https://github.com/xiteen/micROS-cloud
```
- (2) Prepare the base image:
 - 1) Find the dockerfile in the folder: (the path of micROS-cloud)/micROS-cloud/base-image and build it.

```
$ cd ../micROS-cloud/base-image
```

```
$ sudo docker build .
```
 - 2) Get the id of the new created image:

```
$ docker images
```
 - 3) Rename the image:

```
$ sudo docker tag <id> ros:my
```
- (3) Start the micROS- cloud server with the command:

```
$ cd ../micROS-cloud
```

```
$ python run.py
```

Note: If there is an error: Unable to list the containers. Reason: ('Connection aborted.', error(13, 'Permission denied')). Please run the command with the system permissions.

For example: The running of the server.

```
huben@huben: ~/workspace/micROS-cloud
huben@huben:~$ cd workspace/micROS-cloud
huben@huben:~/workspace/micROS-cloud$ sudo python run.py
[sudo] password for huben:
/usr/local/lib/python2.7/dist-packages/flask_sqlalchemy/__init__.py:800: UserWarning: SQLAlchemy_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.
  warnings.warn('SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.')
[11:04:45][INFO] The query of existing containers
[11:04:45][INFO] Remove the container fc5598a67a1f5f303494452560dbb5798449b61932baf9239c99376042eee46d
[11:04:45][INFO] * Running on http://0.0.0.0:5002/ (Press CTRL+C to quit)
[11:04:45][INFO] * Restarting with stat
/usr/local/lib/python2.7/dist-packages/flask_sqlalchemy/__init__.py:800: UserWarning: SQLAlchemy_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.
  warnings.warn('SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.')
[11:04:46][INFO] The query of existing containers
[11:04:46][WARNING] * Debugger is active!
[11:04:46][INFO] * Debugger pin code: 238-318-554
```

Function introduction

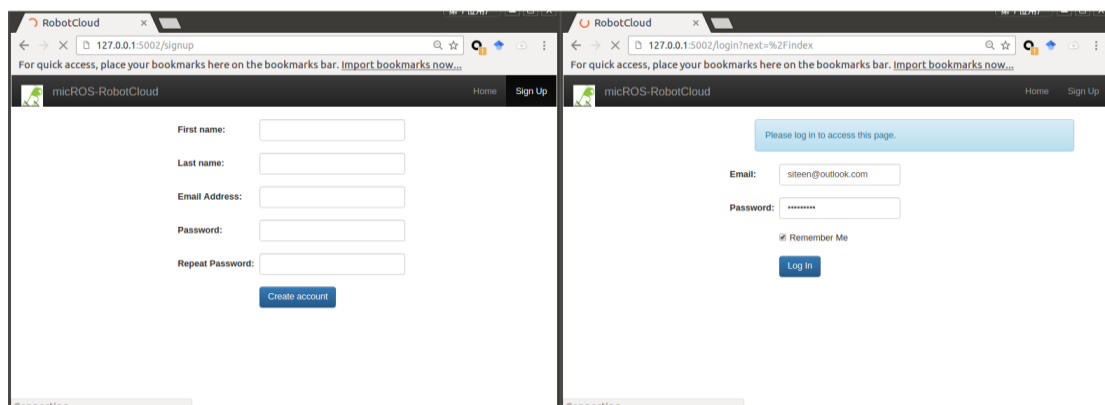
Open the browser and enter the IP address.

<server ip address>:5002

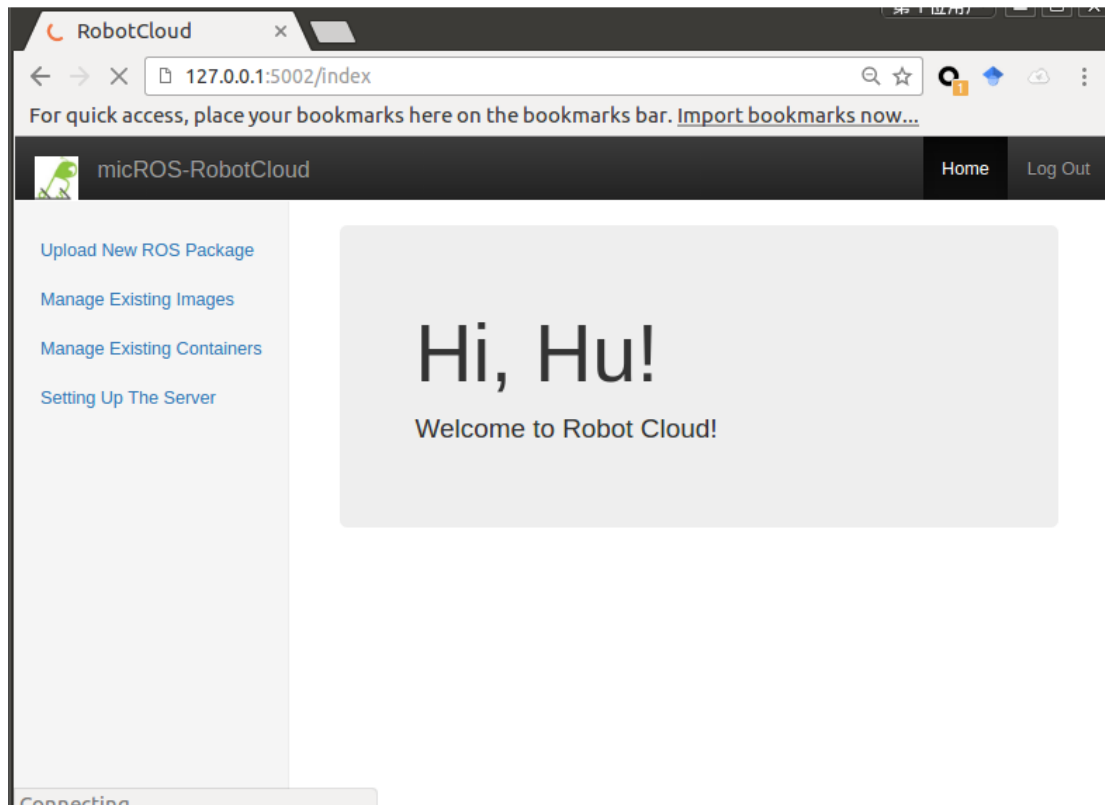
For example: localhost:5002

Registration and login

Click the Sign Up button to create an account and then login:



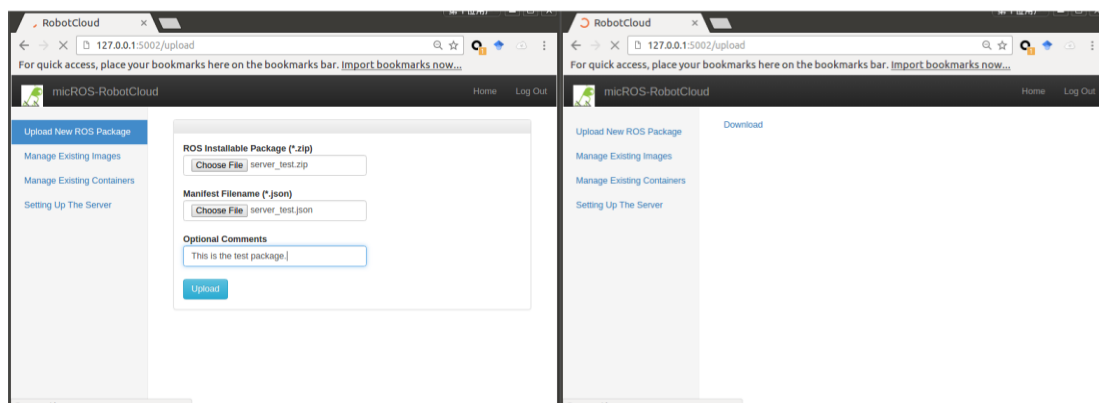
After login, you will see the welcome page:



Upload ROS package

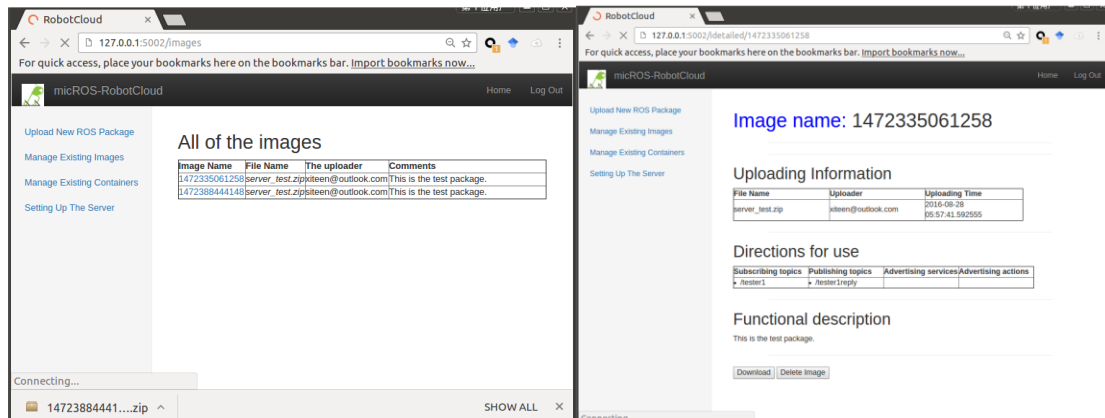
In the *Upload New ROS Package* page, you can upload a ROS package and a json file to define a cloud service. After uploading successfully, you can download a proxy for you robot to access the service.

For example, you can use the ROS package *server_test.zip* and json file *server_test.json* in the */micROS-cloud/test-case/uploadfile* folder.



Manage the existing images

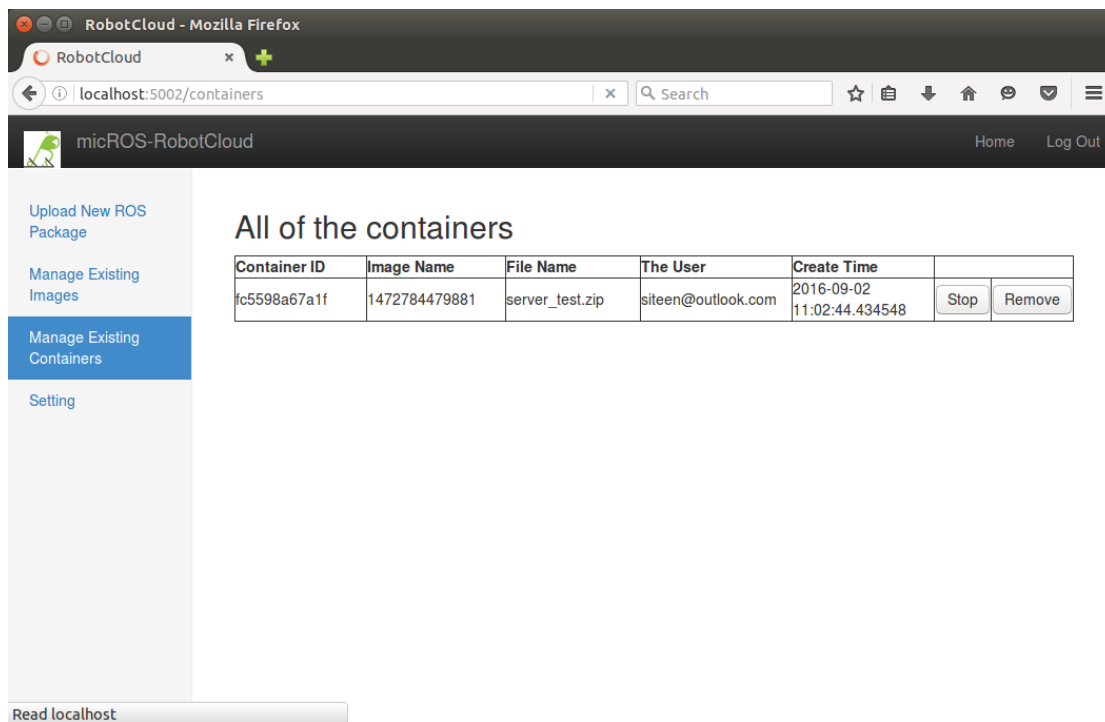
In the *Manage Existing Images* page, you can get the information of every specific docker image.



Click the image name and you can get the details of the image. In this page you can also download the proxy or delete the image.

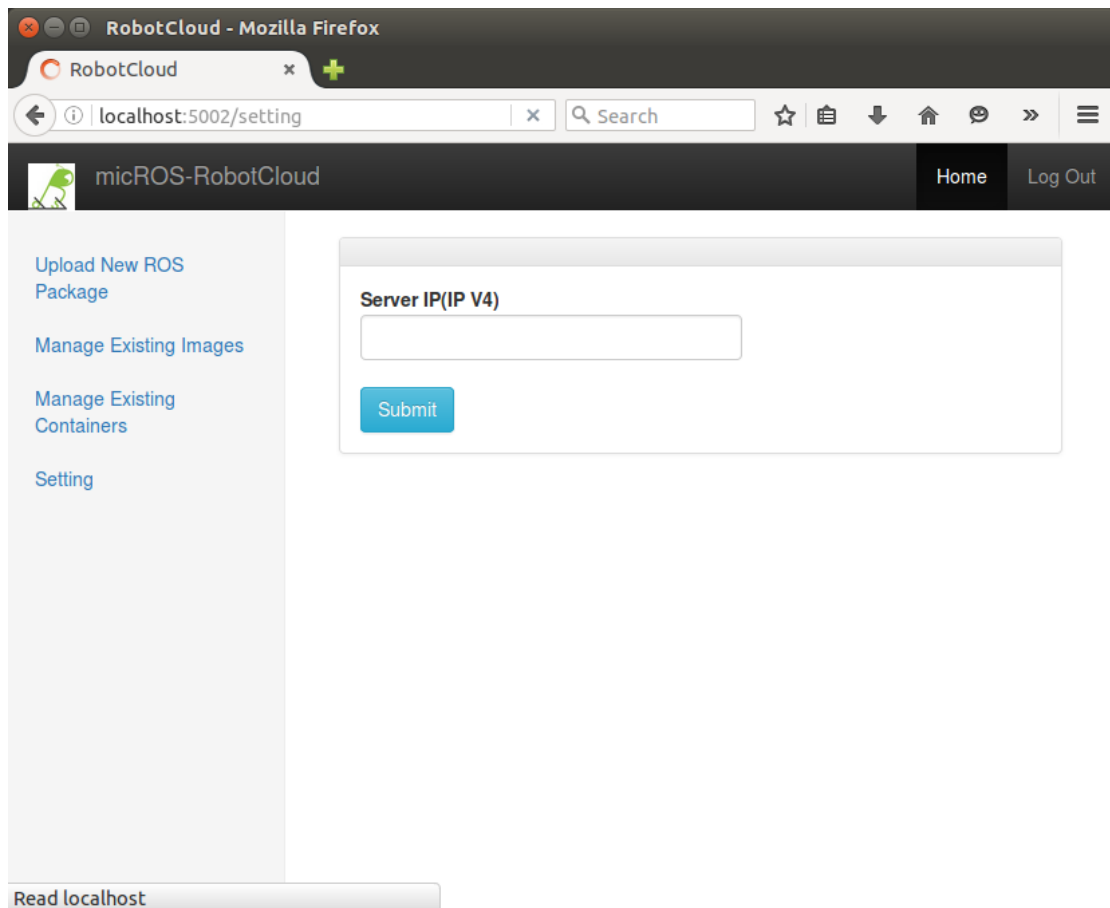
Manage the existing containers

In the *Manage Existing Containers* page, you can get the information of containers and you can stop/start a container or remove a container.



Setting

In the *Setting* page, you can set the server ip address for a remote robot to connect to the service.



micROS-cloud client

Environment

- Python 2.7.6
- ROS indigo

Installation

ROS installation

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

- (1) Follow the tutorial to install the ROS: <http://wiki.ros.org/indigo/Installation/Ubuntu>
- (2) Create a ROS Workspace:
<http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>

micROS-cloud client start

After upload the ROS package you can get the proxy package.

- (1) Download and unzip the proxy package.
- (2) Copy the package to the `~/catkin_ws/src`
- (3) Open the terminal and make the package

```
$ cd catkin_ws
$ catkin_make
```

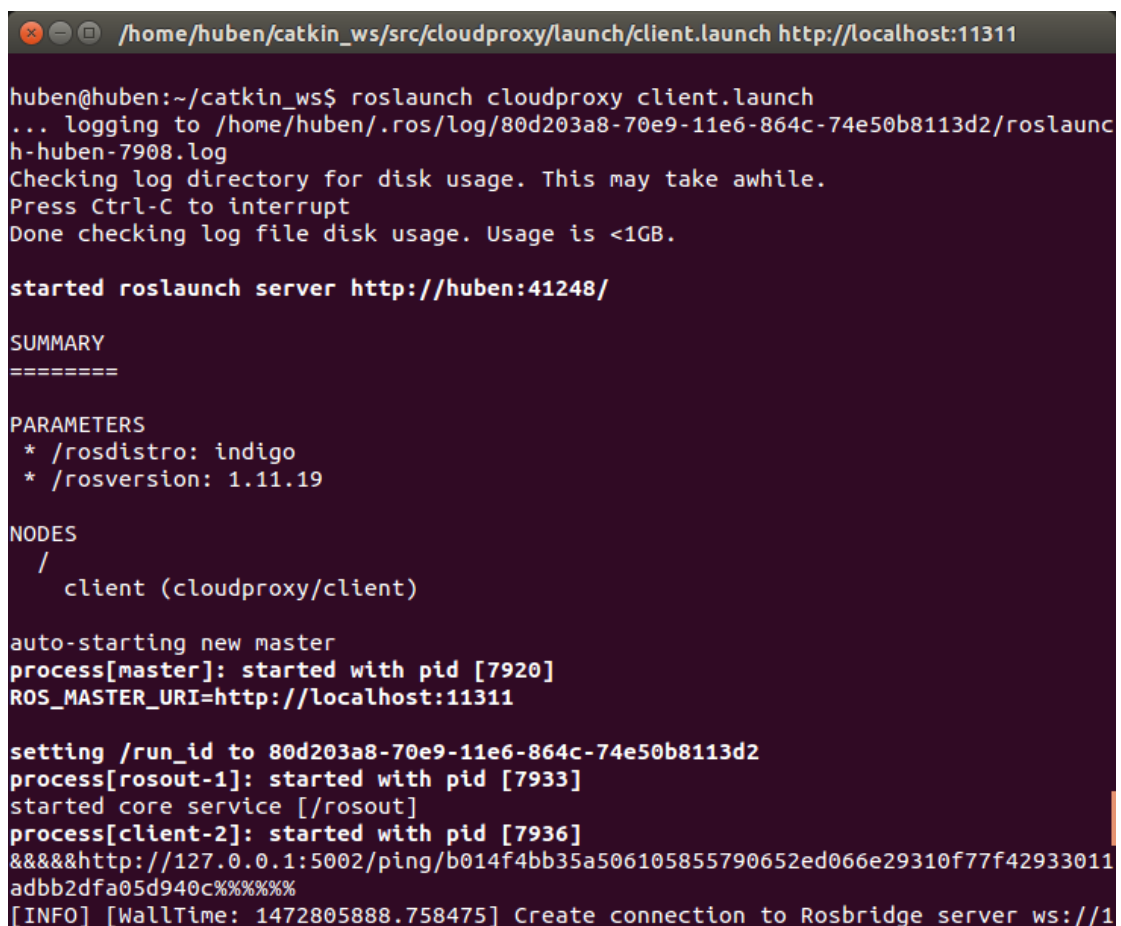
- (4) Run the proxy:

```
$ roslaunch cloudproxy client.launch
```

- (5) After the proxy is running, you can run the other ROS node to publish the topics that the remote running server required.

For example: As the example showed before, you have upload the ROS package *server_test.zip* and json file *server_test.json* which are in the folder `micROS-cloud/test-cast/uploadfile`. If upload successfully, you can download a zip package named with a string of 13 numbers. Deal with the zip package as described before and then run the proxy.

Run the proxy:



```
/home/huben/catkin_ws/src/cloudproxy/launch/client.launch http://localhost:11311

huben@huben:~/catkin_ws$ roslaunch cloudproxy client.launch
... logging to /home/huben/.ros/log/80d203a8-70e9-11e6-864c-74e50b8113d2/roslaunc
h-huben-7908.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://huben:41248/

SUMMARY
=====

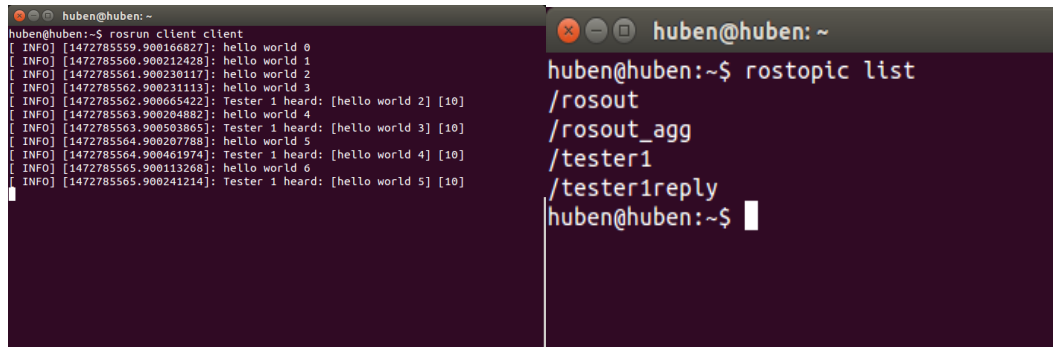
PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.19

NODES
/
  client (cloudproxy/client)

auto-starting new master
process[master]: started with pid [7920]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 80d203a8-70e9-11e6-864c-74e50b8113d2
process[rosout-1]: started with pid [7933]
started core service [/rosout]
process[client-2]: started with pid [7936]
&&&&http://127.0.0.1:5002/ping/b014f4bb35a506105855790652ed066e29310f77f42933011
adbb2dfa05d940c%
[INFO] [WallTime: 1472805888.758475] Create connection to Rosbridge server ws://1
```

After the proxy is running, it is waiting for the necessary topics from the local. Start the ROS node *client* (You can find it in `/micROS-cloud/test-case/local`, copy the package *src-test.zip* to `catkin_ws` and unzip the package and then `catkin_make` them) to publish the topic that the proxy get it and send to the server.



The image shows two terminal windows side-by-side. The left window, titled 'huben@huben: ~', shows the output of the command 'roslaunch client client'. It displays a series of 'hello world' messages from the client and 'Tester 1 heard' messages from the server, indicating successful communication. The right window, also titled 'huben@huben: ~', shows the output of the command 'rostopic list', which lists the topics: '/rosout', '/rosout_agg', '/tester1', and '/tester1reply'.

```
huben@huben: ~  
huben@huben:~$ roslaunch client client  
[ INFO] [1472785559.900166827]: hello world 0  
[ INFO] [1472785560.900212428]: hello world 1  
[ INFO] [1472785561.900230117]: hello world 2  
[ INFO] [1472785562.900231113]: hello world 3  
[ INFO] [1472785562.900665422]: Tester 1 heard: [hello world 2] [10]  
[ INFO] [1472785563.900204882]: hello world 4  
[ INFO] [1472785563.900503865]: Tester 1 heard: [hello world 3] [10]  
[ INFO] [1472785564.900207788]: hello world 5  
[ INFO] [1472785564.900461974]: Tester 1 heard: [hello world 4] [10]  
[ INFO] [1472785565.900113268]: hello world 6  
[ INFO] [1472785565.900241214]: Tester 1 heard: [hello world 5] [10]  
  
huben@huben: ~  
huben@huben:~$ rostopic list  
/rosout  
/rosout_agg  
/tester1  
/tester1reply  
huben@huben:~$
```

As you can see, the node client published the node /tester1 and received topic /tester1reply what is published by server and get from the server by proxy.