

Chapter 2 Getting Started with ROS Programming

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Chapter 2 dari buku "Mastering ROS for Robotics Programming" membahas tentang memulai pemrograman ROS. Setelah membahas dasar-dasar ROS master, parameter server, dan roscore, kita dapat mulai membuat dan membangun ros packages. Pada bab ini, kita akan membuat berbagai node ROS dengan mengimplementasikan sistem komunikasi ROS. Selama bekerja dengan ros packages, kita juga akan memperbarui pengetahuan dasar tentang node ROS, topik, pesan, layanan, dan actionlib.

Creating ROS Packages

Ros packages merupakan unit dasar dari program ROS. Paket ini dapat dibuat, dikompilasi, dan dirilis ke publik. ROS saat ini yang digunakan adalah Noetic Ninjemys dengan sistem pembangunan catkin. Sistem ini bertanggung jawab untuk menghasilkan target (eksekutabel/libraries) dari kode sumber teks yang dapat digunakan oleh pengguna akhir. Dalam distribusi ROS sebelumnya seperti Electric dan Fuerte, menggunakan sistem pembangunan rosbld. Namun, karena kelemahan rosbld, catkin muncul dan memungkinkan sistem kompilasi ROS mendekati Cross Platform Make (CMake). Langkah-langkah mencakup pembuatan workspace catkin, inisialisasi workspace, dan kompilasi workspace menggunakan perintah catkin_make.

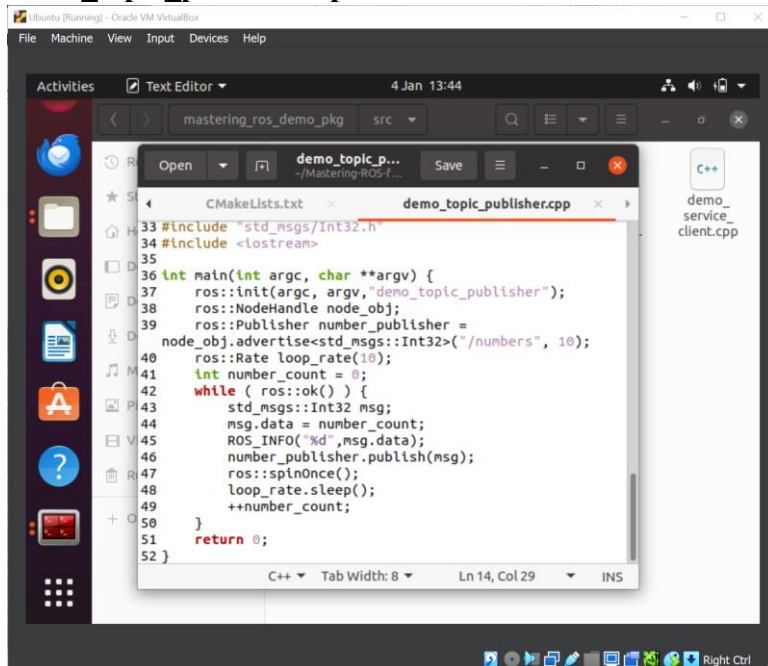
Working with ROS Topics

Topik ROS digunakan sebagai metode komunikasi antara node ROS, memungkinkan mereka berbagi aliran informasi yang dapat diterima oleh node lain. Pembahasan di bagian ini fokus pada pembuatan dua node ROS untuk mempublikasikan dan berlangganan topik. Setelah membuat workspace catkin, kita dapat membuat ros packages menggunakan perintah catkin_create_pkg. Dalam contoh ini, paket mastering_ros_demo_pkg dibuat dengan dependensi roscpp, std_msgs, actionlib, dan actionlib_msgs. Setelah pembuatan paket, dependencies tambahan dapat ditambahkan manual melalui pengeditan file CMakeLists.txt dan package.xml. Dijelaskan pula cara membangun paket dan mulai menambahkan node ke folder src dalam paket tersebut. Setelah berhasil dikompilasi, pembahasan beralih ke cara bekerja dengan ROS topics menggunakan dua file kode sumber demo_topic_publisher.cpp dan demo_topic_subscriber.cpp.

Creating ROS nodes

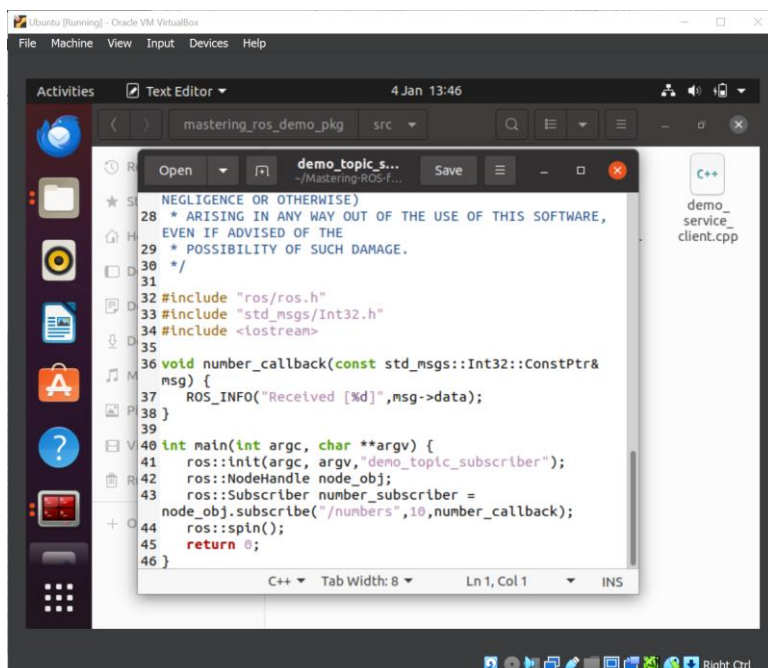
Node pertama yang dibahas adalah `demo_topic_publisher.cpp`. Node ini akan mempublikasikan nilai integer pada topik bernama `/numbers`. Kode ini menggambarkan penggunaan header files, inisialisasi ROS node, pembuatan node handle, pembuatan publisher untuk topik `/numbers`, dan perulangan tak terbatas yang mengirimkan pesan integer ke topik tersebut dengan kecepatan 10 Hz. Setelah itu, dibahas juga node subscriber `demo_topic_subscriber.cpp` yang berlangganan topik `/numbers` dan memiliki callback function untuk menangani pesan yang diterima.

demo_topic_publisher.cpp



```
1 #include <std_msgs/Int32.h>
2 #include <iostream>
3
4 int main(int argc, char **argv) {
5     ros::init(argc, argv, "demo_topic_publisher");
6     ros::NodeHandle node_obj;
7     ros::Publisher number_publisher =
8         node_obj.advertise<std_msgs::Int32>("/numbers", 10);
9     ros::Rate loop_rate(10);
10    int number_count = 0;
11    while ( ros::ok() ) {
12        std_msgs::Int32 msg;
13        msg.data = number_count;
14        ROS_INFO("%d", msg.data);
15        number_publisher.publish(msg);
16        ros::spinOnce();
17        loop_rate.sleep();
18        ++number_count;
19    }
20    return 0;
21 }
```

demo_topic_subscriber.cpp

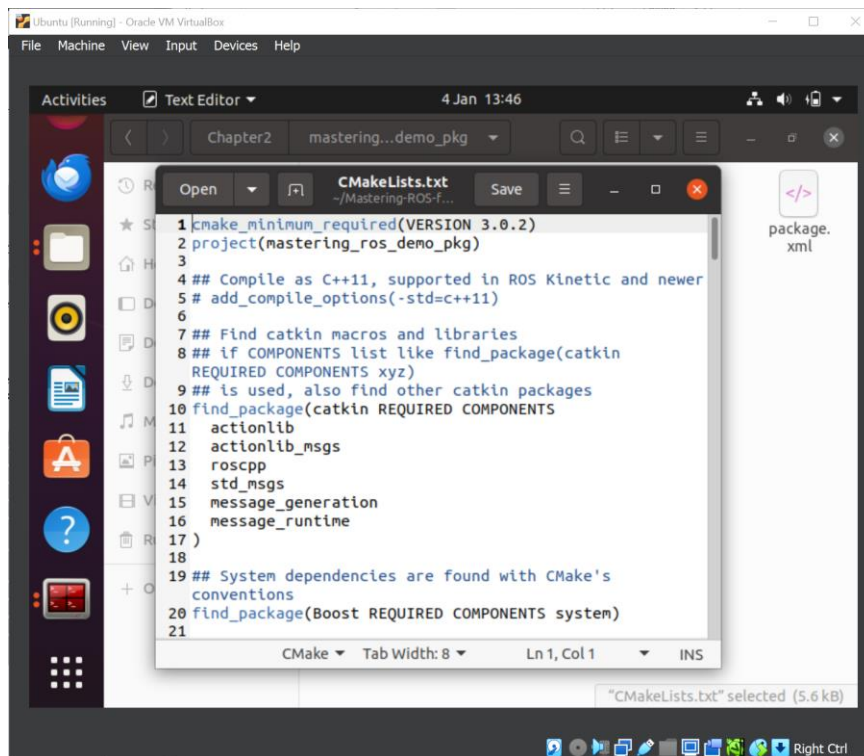


```
1 // NEGLIGENCE OR OTHERWISE
2 // ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
3 // EVEN IF ADVISED OF THE
4 // POSSIBILITY OF SUCH DAMAGE.
5
6 #include "ros/ros.h"
7 #include "std_msgs/Int32.h"
8 #include <iostream>
9
10 void number_callback(const std_msgs::Int32::ConstPtr&
11 msg) {
12     ROS_INFO("Received [%d]", msg->data);
13 }
14
15 int main(int argc, char **argv) {
16     ros::init(argc, argv, "demo_topic_subscriber");
17     ros::NodeHandle node_obj;
18     ros::Subscriber number_subscriber =
19         node_obj.subscribe("/numbers", 10, number_callback);
20     ros::spin();
21     return 0;
22 }
```

Building the nodes

Proses ini melibatkan penyuntingan file CMakeLists.txt dalam paket untuk mengkompilasi dan membangun kode sumber. Kode CMakeLists.txt mengarahkan pada file-file yang perlu dikompilasi dan dihubungkan. Setelah penyuntingan, perintah catkin_make digunakan untuk membangun workspace ROS dan menjalankan kedua node tersebut. Diagram komunikasi antara node penerbit dan pelanggan ditunjukkan, dan alat ROS seperti rosnode dan rostopic digunakan untuk debug dan pemahaman kerja kedua node.

CMakeLists.txt

A screenshot of a text editor window titled 'CMakeLists.txt' showing the content of a CMake file. The editor is running on Ubuntu in an Oracle VM VirtualBox. The file content is as follows:

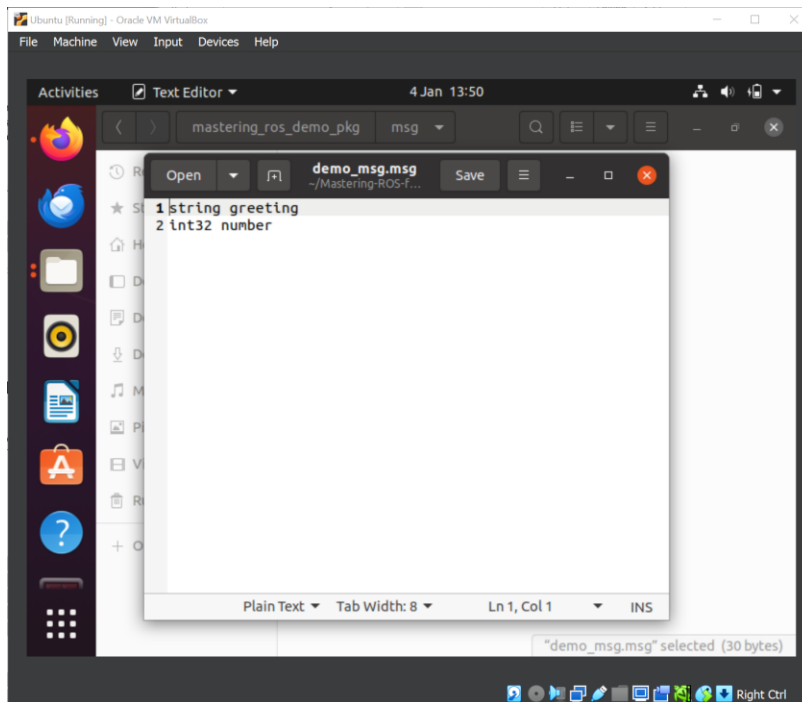
```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(mastering_ros_deno_pkg)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin
   REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   actionlib
12   actionlib_msgs
13   roscpp
14   std_msgs
15   message_generation
16   message_runtime
17 )
18
19 ## System dependencies are found with CMake's
   conventions
20 find_package(Boost REQUIRED COMPONENTS system)
21
```

The editor interface includes a sidebar with file explorer icons, a top menu bar with 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help', and a status bar at the bottom showing 'CMake', 'Tab Width: 8', 'Ln 1, Col 1', and 'INS'. The window title bar indicates 'Ubuntu [Running] - Oracle VM VirtualBox'.

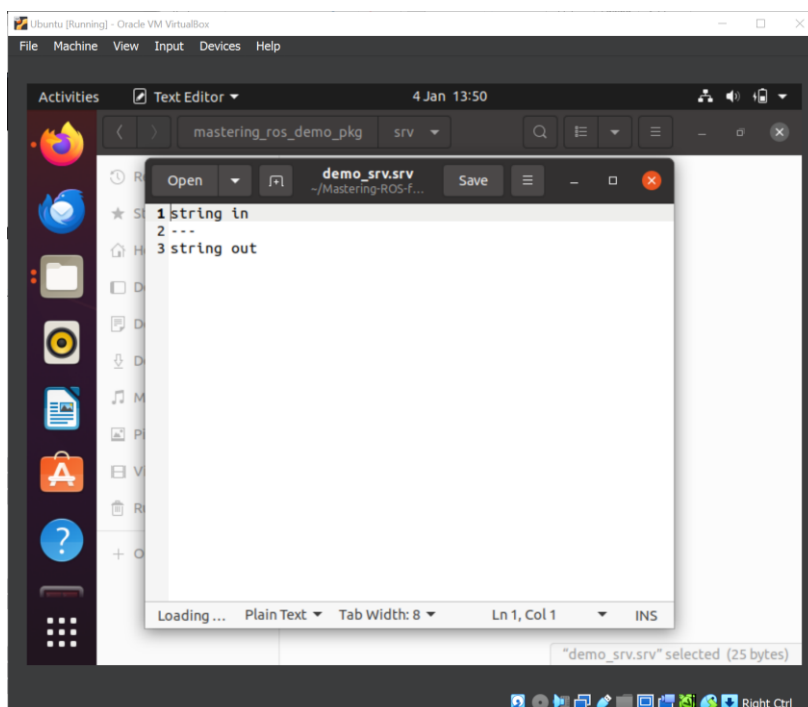
Adding custom .msg and .srv files

Bagian ini membahas pembuatan pesan (message) dan definisi layanan (service) kustom dalam ros packages. Pesan disimpan dalam berkas .msg, sementara definisi layanan disimpan dalam berkas .srv. Kedua definisi ini memberitahu ROS tentang jenis data dan nama data yang akan dikirim dari sebuah node ROS. Saat pesan kustom ditambahkan, ROS akan mengonversi definisi tersebut menjadi kode C++ yang dapat diikutsertakan dalam node-node kita.

demo_msg.msg



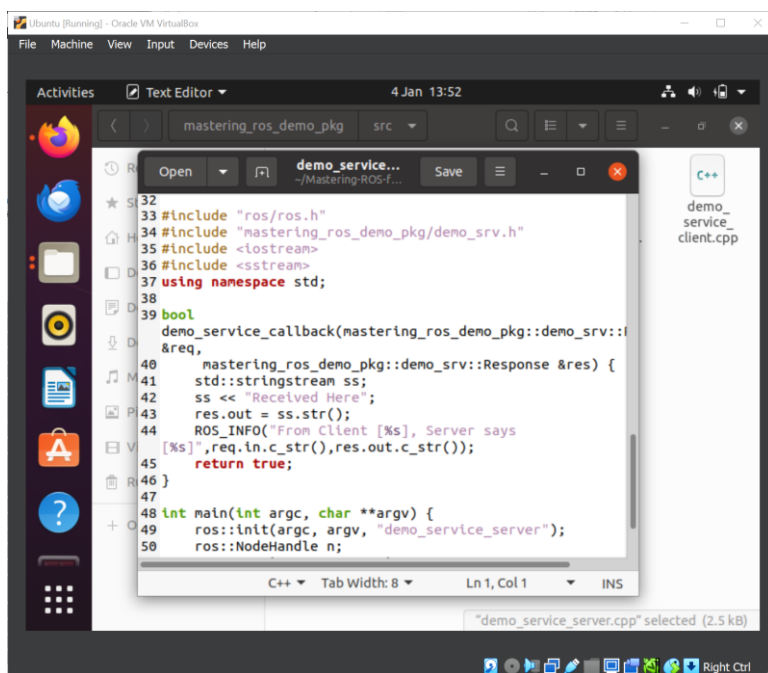
demo_srv.srv



Working with ROS services

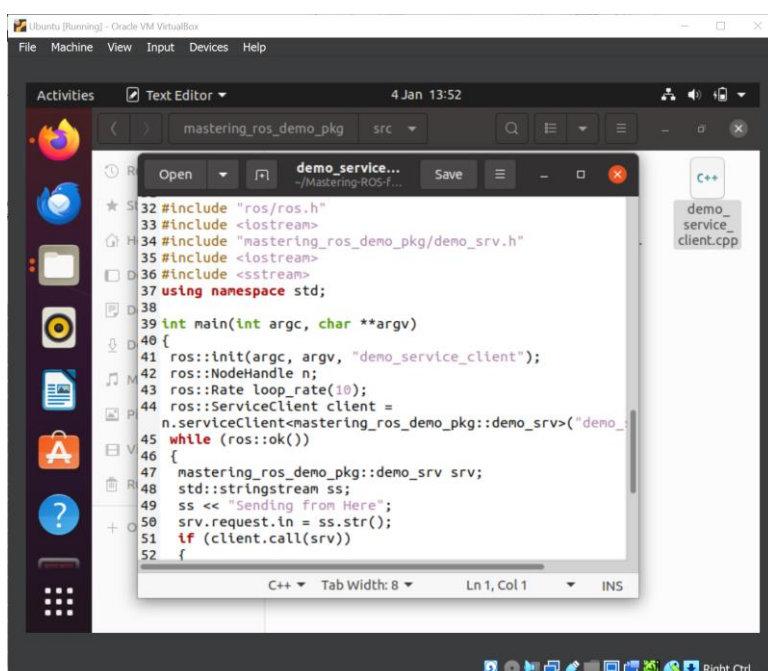
- Membuat node ROS yang menggunakan definisi layanan yang sudah ditentukan sebelumnya.
- Node layanan dapat mengirimkan pesan string sebagai permintaan ke server, dan server akan mengirimkan pesan lain sebagai respons.
- Dua node yang dibuat adalah `demo_service_server.cpp` (server) dan `demo_service_client.cpp` (klien).

demo_service_server.cpp



```
32 #include "ros/ros.h"
33 #include "mastering_ros_demo_pkg/demo_srv.h"
34 #include <iostream>
35 #include <sstream>
36 #include <string>
37 using namespace std;
38
39 bool
demo_service_callback(mastering_ros_demo_pkg::demo_srv::Request
&req,
40 mastering_ros_demo_pkg::demo_srv::Response &res) {
41     std::stringstream ss;
42     ss << "Received Here";
43     res.out = ss.str();
44     ROS_INFO("From Client [%s], Server says [%s]", req.in.c_str(), res.out.c_str());
45     return true;
46 }
47
48 int main(int argc, char **argv) {
49     ros::init(argc, argv, "demo_service_server");
50     ros::NodeHandle n;
```

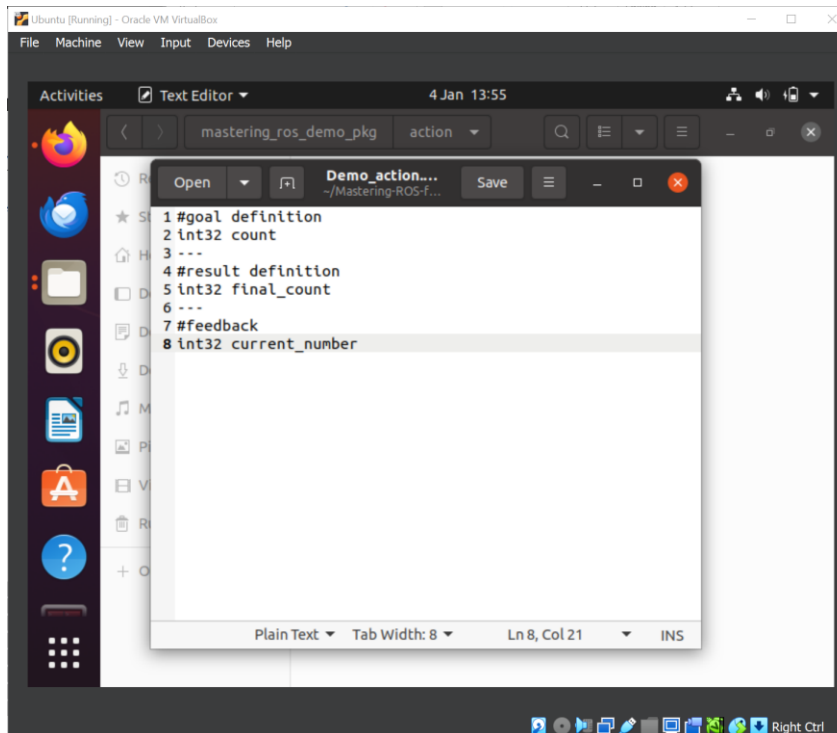
demo_service_client.cpp



```
32 #include "ros/ros.h"
33 #include <iostream>
34 #include "mastering_ros_demo_pkg/demo_srv.h"
35 #include <iostream>
36 #include <sstream>
37 using namespace std;
38
39 int main(int argc, char **argv)
40 {
41     ros::init(argc, argv, "demo_service_client");
42     ros::NodeHandle n;
43     ros::Rate loop_rate(10);
44     ros::ServiceClient client =
n.serviceClient<mastering_ros_demo_pkg::demo_srv>("demo_
45     while (ros::ok())
46     {
47         mastering_ros_demo_pkg::demo_srv srv;
48         std::stringstream ss;
49         ss << "Sending From Here";
50         srv.request.in = ss.str();
51         if (client.call(srv))
52         {
```

Working with ROS actionlib

Demo_action.action

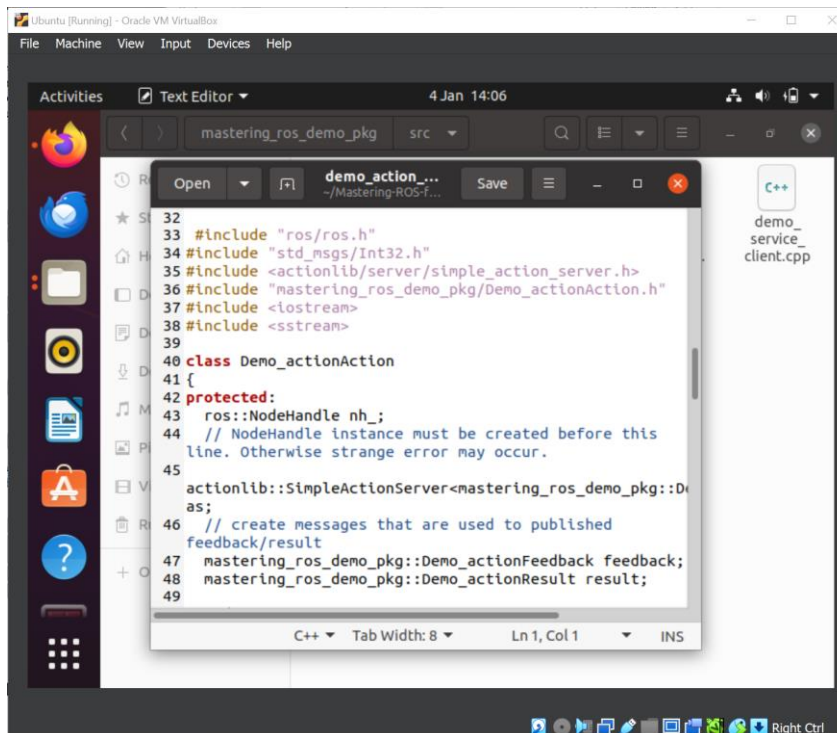


A screenshot of a text editor window titled "Demo_action..." showing the content of a ROS action file. The editor is open in a virtual machine environment (Ubuntu [Running] - Oracle VM VirtualBox). The file content is as follows:

```
1 #goal definition
2 int32 count
3 ---
4 #result definition
5 int32 final_count
6 ---
7 #feedback
8 int32 current_number
```

Creating the ROS action server

demo_action_server.cpp

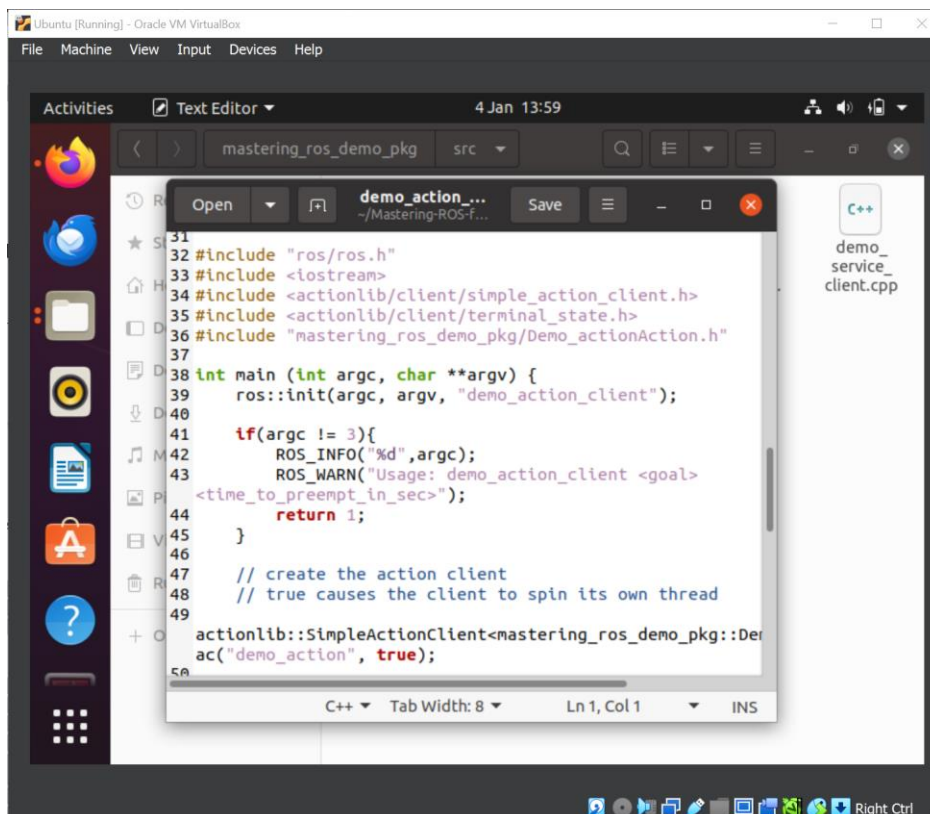


A screenshot of a text editor window titled "demo_action..." showing the content of a C++ source file. The editor is open in a virtual machine environment (Ubuntu [Running] - Oracle VM VirtualBox). The file content is as follows:

```
32 #include "ros/ros.h"
33 #include "std_msgs/Int32.h"
34 #include <actionlib/server/simple_action_server.h>
35 #include "mastering_ros_demo_pkg/Demo_actionAction.h"
36 #include <iostream>
37 #include <sstream>
38 #include <sstream>
39
40 class Demo_actionAction
41 {
42 protected:
43   ros::NodeHandle nh_;
44   // NodeHandle instance must be created before this
   line. Otherwise strange error may occur.
45   actionlib::SimpleActionServer<mastering_ros_demo_pkg::D
   as;
46   // create messages that are used to published
   feedback/result
47   mastering_ros_demo_pkg::Demo_actionFeedback feedback;
48   mastering_ros_demo_pkg::Demo_actionResult result;
49
```

- Action server menerima nilai tujuan dalam bentuk angka dan akan menghitung dari 0 hingga nilai tersebut.
- Jika perhitungan selesai, server mengirimkan hasilnya; jika tidak, tugas dapat dicabut oleh klien.
- Implementasi menggunakan actionlib untuk dapat membatalkan tugas yang berjalan dan memulai tugas baru jika diperlukan.
- Contoh tugas action server adalah demo_action_server.cpp.

demo_action_client.Cpp



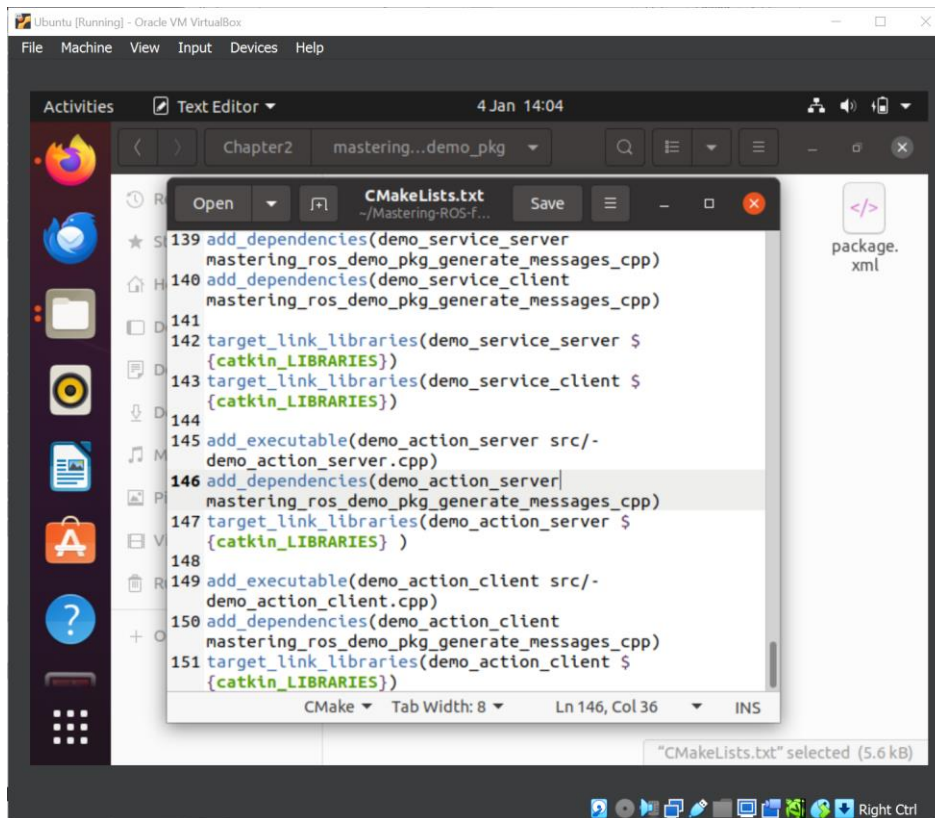
```

31
32 #include "ros/ros.h"
33 #include <iostream>
34 #include <actionlib/client/simple_action_client.h>
35 #include <actionlib/client/terminal_state.h>
36 #include "mastering_ros_demo_pkg/Demo_actionAction.h"
37
38 int main (int argc, char **argv) {
39     ros::init(argc, argv, "demo_action_client");
40
41     if(argc != 3){
42         ROS_INFO("%d",argc);
43         ROS_WARN("Usage: demo_action_client <goal>
44         <time_to_preempt_in_sec>");
45         return 1;
46     }
47     // create the action client
48     // true causes the client to spin its own thread
49     actionlib::SimpleActionClient<mastering_ros_demo_pkg::Demo_actionAction> ac("demo_action", true);
50

```

- Action client mengirim nilai tujuan ke server dan menunggu hingga batas waktu yang ditentukan.
- Jika tugas selesai, client menerima hasilnya; jika tidak, client dapat membatalkan tugas.
- Contoh implementasi adalah demo_action_client.cpp.

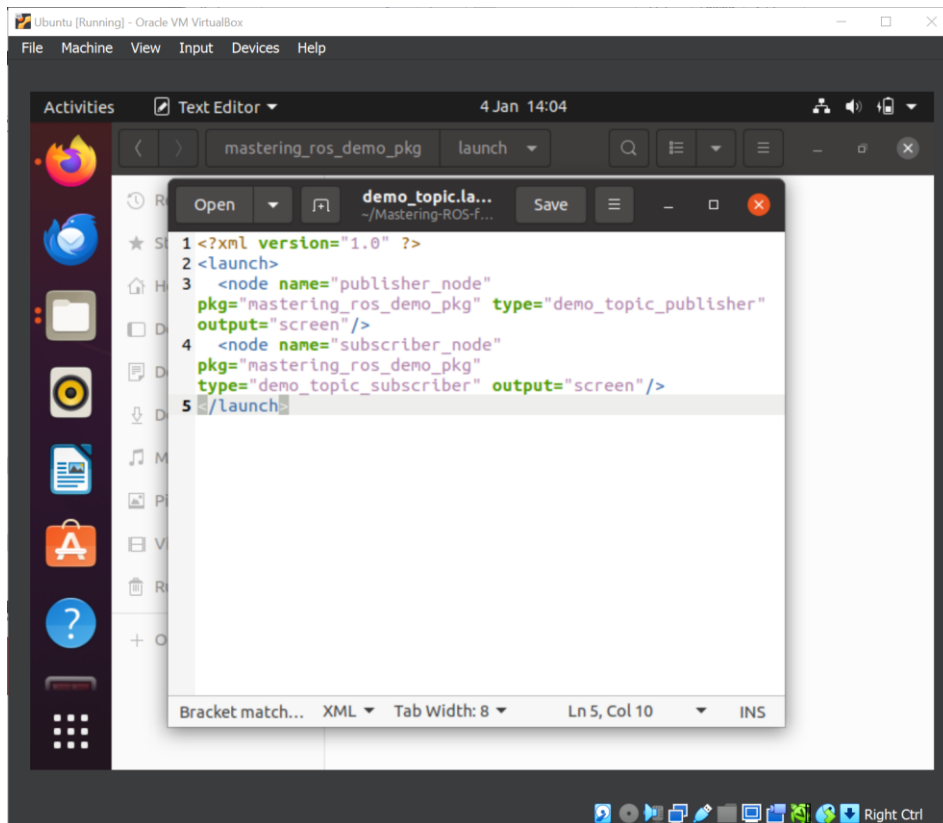
Building the ROS action server and client

A screenshot of a text editor window titled 'CMakeLists.txt' showing CMake code. The code includes dependencies for 'demo_service_server', 'demo_service_client', 'demo_action_server', and 'demo_action_client', all linking to 'catkin_LIBRARIES'. The editor interface includes a sidebar with file explorer, a top bar with search and menu icons, and a status bar at the bottom indicating 'CMake', 'Tab Width: 8', 'Ln 146, Col 36', and 'INS' mode. The file 'package.xml' is also visible in the sidebar.

```
139 add_dependencies(demo_service_server
mastering_ros_demo_pkg_generate_messages_cpp)
140 add_dependencies(demo_service_client
mastering_ros_demo_pkg_generate_messages_cpp)
141
142 target_link_libraries(demo_service_server $
{catkin_LIBRARIES})
143 target_link_libraries(demo_service_client $
{catkin_LIBRARIES})
144
145 add_executable(demo_action_server src/-
demo_action_server.cpp)
146 add_dependencies(demo_action_server
mastering_ros_demo_pkg_generate_messages_cpp)
147 target_link_libraries(demo_action_server $
{catkin_LIBRARIES} )
148
149 add_executable(demo_action_client src/-
demo_action_client.cpp)
150 add_dependencies(demo_action_client
mastering_ros_demo_pkg_generate_messages_cpp)
151 target_link_libraries(demo_action_client $
{catkin_LIBRARIES})
```

- Package.xml dan CMakeLists.txt diubah untuk memasukkan dependensi dan mengonfigurasi pembangunan tiga paket actionlib, actionlib_msgs, dan message_generation.
- Penggunaan Boost sebagai dependensi sistem.
- Dua file action ditambahkan ke folder action untuk mendefinisikan spesifikasi aksi.
- Dua executable (action server dan action client) ditambahkan ke CMakeLists.txt dengan dependensi yang sesuai.
- Setelah catkin_make, nodes dapat dijalankan menggunakan perintah rosrun dan diuji dengan mrobservice dan rostopic.

Creating launch files



- Launch files (launch files) di ROS sangat berguna untuk menjalankan lebih dari satu node sekaligus.
- Dengan launch files, kita dapat menjalankan beberapa node ROS secara bersamaan tanpa harus membuka terminal untuk setiap node.
- Launch files adalah berkas berbasis XML yang berisi definisi node-node yang akan dijalankan.
- Perintah roslaunch secara otomatis memulai ROS master dan parameter server, menghilangkan kebutuhan untuk menjalankan roscore dan node-node secara terpisah.
- Launch files dapat memudahkan pengelolaan node-node yang kompleks, seperti pada kasus robot dengan puluhan node.
- Launch files ditempatkan dalam folder "launch" di dalam ros packages.
- Isi launch files (contoh: demo_topic.launch) terdiri dari elemen-elemen seperti <node> yang mendefinisikan node-node yang akan dijalankan.
- Setelah membuat launch files, dapat diluncurkan dengan perintah roslaunch dan memuat semua node yang dijelaskan di dalamnya.
- Informasi debug dan log dapat dilihat menggunakan perintah rosnode list dan rqt_console.
- Dengan menggunakan launch files, proses menjalankan dan mengelola node-node ROS menjadi lebih efisien dan terorganisir.

