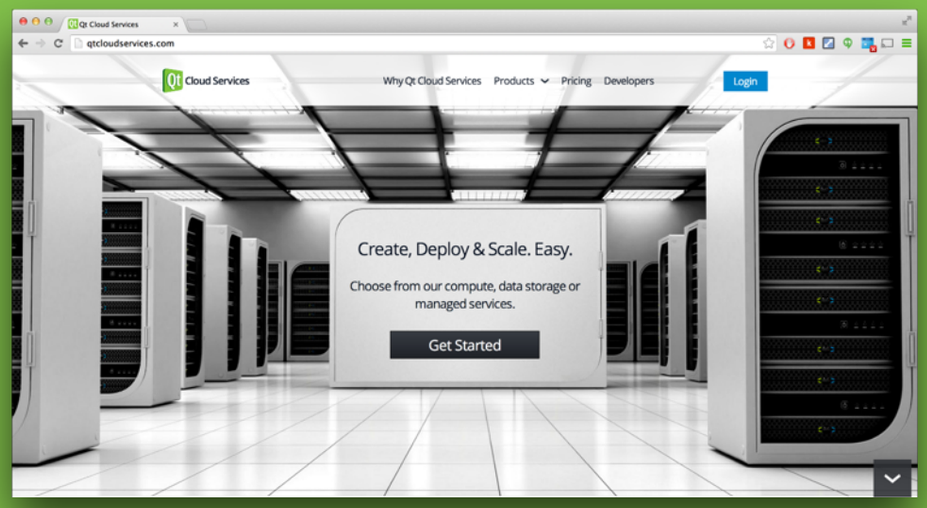
Give a Push to Your Qt Application with Qt Cloud Services and WebSockets

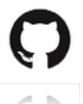


Qt Cloud Services

About me

Qt Cloud Services

Lauri Nevala

 nevalau  nevalla  nevalla

- Working in Qt Cloud Services team at The Qt Company
- Over ten years of experience in creating web and mobile based applications

Qt Cloud Services

Our Goal

Understand how to use Qt Cloud Services in order to send and receive WebSocket messages

Qt Cloud Services

Qt Cloud Services

Qt Cloud Services

What is Qt Cloud Services

Enginio Data Storage
Storage for Application Data
Use with QtEnginio Library

Managed Application Runtime
Application Platform as a Service
Support for Qt/C++, NodeJS, Apache, PHP, MongoDB, MySQL, Redis…

Managed WebSocket
Real-Time Socket Connections with virtually unlimited scalability
Use with SDK’s and Qt WebSocket Client Library

Enginio Data Storage (EDS)

Flexible and powerful
cloud data storage
with built-in user and
data access control

Qt Cloud Services

Managed Application Runtime (MAR)

Scalable,
Multi-language,
Multi-database,
Application Platform as a Service

Qt Cloud Services

Managed Application Runtimes

How does it work?

Qt Cloud Services

Supported Frameworks

Supported frameworks by 3rd party build packs

Scala, Clojure, Play, Gradle, Grails, PHP, Go, Meteorite, Perl, Dart, Nginx, Apache, Jekyll

Qt Cloud Services

Built-in Services

or choose from our cloud based services

Enginio Data Storage
Managed WebSocket

or choose anything with SDK

... Amazon, Azure, Google ...

Developer Friendly Deployment

Deploy using Git – the most common VCS among developers

```
> git push qtc master
```

Managed WebSocket (MWS)

Fully managed service implementing a bi-directional, real-time communication gateway for WebSockets.

Qt Cloud Services

Managed WebSocket

How does it work?

Qt Cloud Services

3. send WebSocket message

4. deliver WebSocket message

1. get WebSocket URI

2. open WebSocket connection

```
{ data: '{
    "object":{
       "id":"541df1c1e5bde554a805b213",
       "createdAt":"2014-09 20T21:29:37.849Z",
       "device":"FZXC0wg0LvaJ",
       "done":false,
       "objectType":"objects.todos",
       "text":"testi2",
       "updatedAt":"2014-09-20T21:29:37.849Z",
       "userId":"user:54196f3f5a3d8b61860381a4"
    },
    "meta":{
        "backendId":"5417e5485a3d8b418a01adb7",
        "eventName":"create",
        "requestId":"a671f67b4f41c20b53988a6057b32539",
        "userId":null}
    },
    receivers: {
        sockets: ['*'],
        tags: []
    }
}
```

Qt Cloud Services

WebSockets

Consider using WebSockets for instantaneous data without the browser refresh

Qt Cloud Services

Where to use?

- Chats
- Social feeds
- Multiplayer games
- Collaborative editing
- Sport updates
- Location based apps
- Your killer app

Qt Cloud Services

Why WebSockets are so great?

- WebSockets defines an API establishing “socket” connections between web clients and a server
- TCP-based protocol
- Full-duplex, simultaneous bi-directional messaging
- Uri scheme ws: and wss:

Qt Cloud Services

WebSockets and Qt Cloud Services

1. 🗄 + 🔀

   ➡ public socket

2. 🧊 + 🗄 + 🔀

   ➡ private socket

3. 🧊

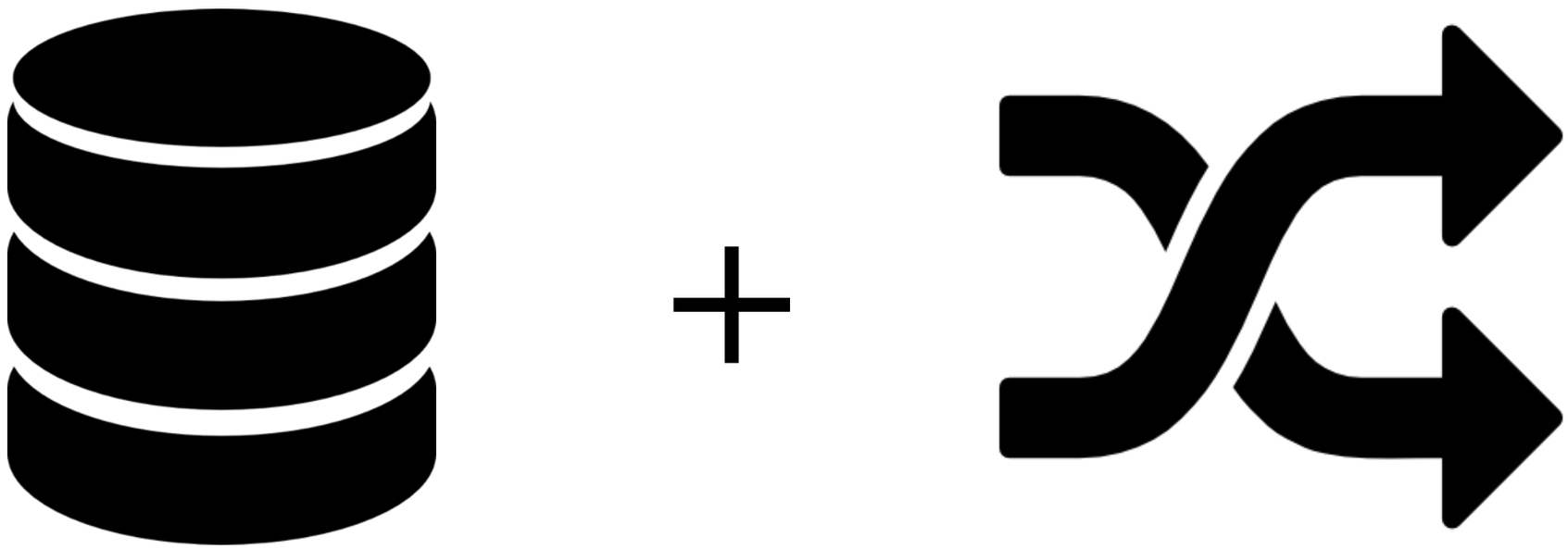
   ➡ do anything you want

Qt Cloud Services

https://flic.kr/p/dUtbk5

Public sockets

Qt Cloud Services



Qt Cloud Services

WebSocket server

- Create EDS instance
- Create MWS instance
- Create EDS Webhook

Qt Cloud Services

Launch new EDS instance

Name

Enter instance name

Datacenter

eds-eu-1

Cancel

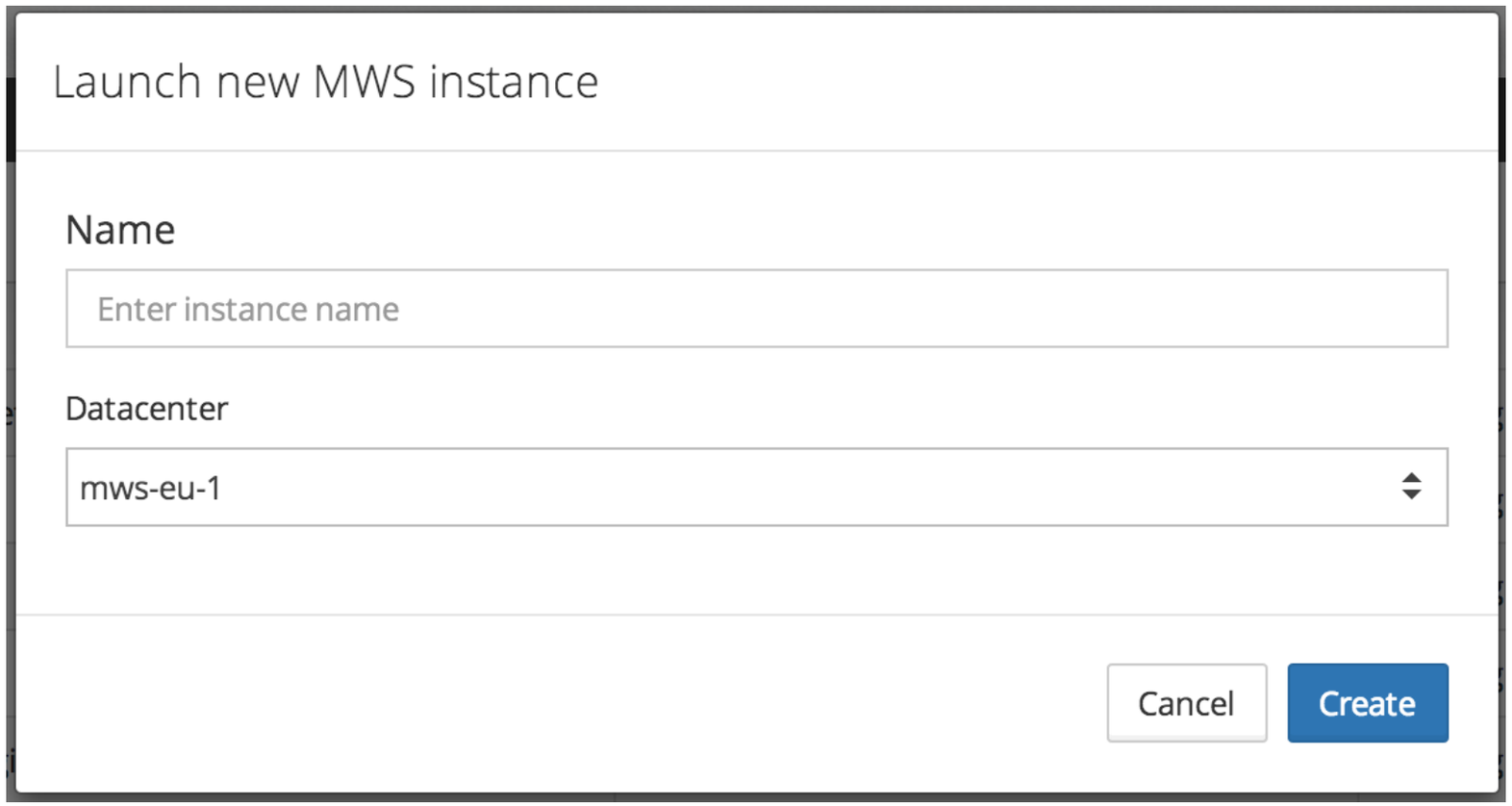
Create

Qt Cloud Services

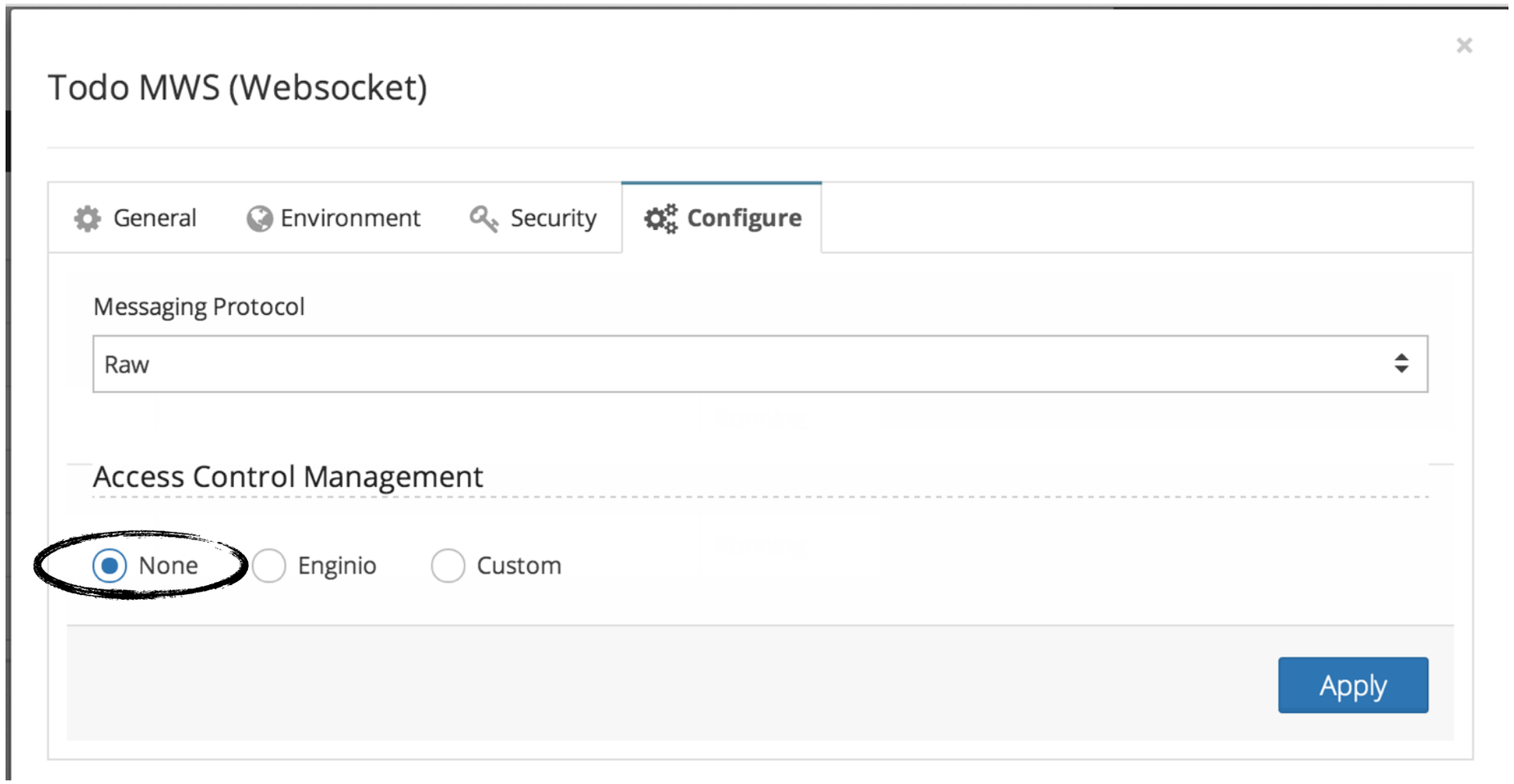
WebSocket server

- [x] Create EDS instance
- [ ] Create MWS instance
- [ ] Create EDS Webhook

Qt Cloud Services



Qt Cloud Services



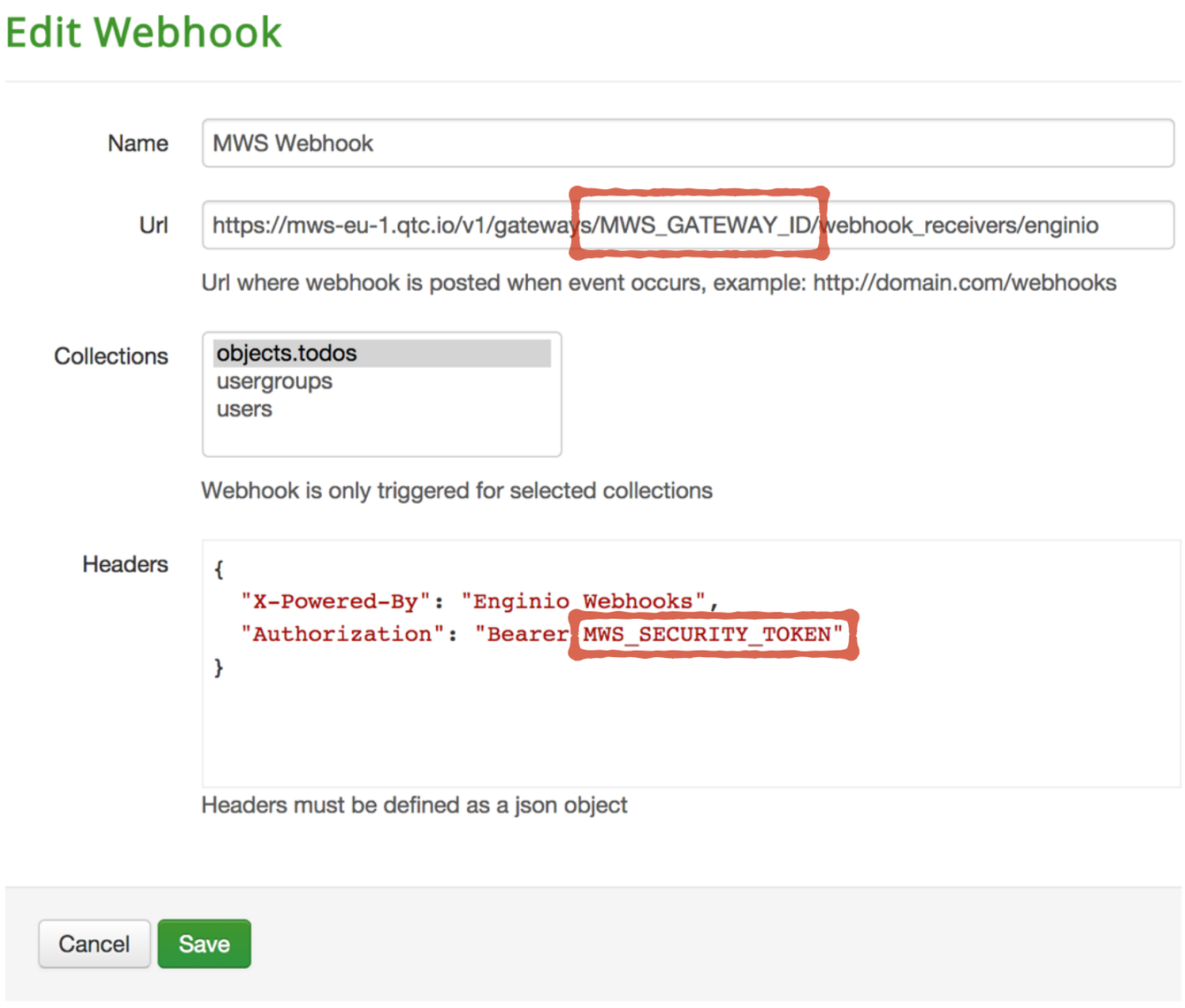
Qt Cloud Services

WebSocket server

- [x] Create EDS instance
- [x] Create MWS instance
- [ ] Create EDS Webhook

Qt Cloud Services

Create EDS Webhook



http://bit.ly/eds-mws-webhook

Qt Cloud Services

EDS

MWS

App

HTTP POST

https://mws-eu-1.qtc.io/v1/gateways/MWS_GATEWAY_ID/webhook_receivers/enginio

```
{ data: '{
    "object":{
       "id":"541df1c1e5bde554a805b213",
       "createdAt":"2014-09 20T21:29:37.849Z",
       "device":"FZXC0wg0LvaJ",
       "done":false,
       "objectType":"objects.todos",
       "text":"testi2",
       "updatedAt":"2014-09-20T21:29:37.849Z",
       "userId":"user:54196f3f5a3d8b61860381a4"
    },
    "meta":{
        "backendId":"5417e5485a3d8b418a01adb7",
        "eventName":"create",
        "requestId":"a671f67b4f41c20b53988a6057b32539",
        "userId":null}
    }'
    receivers: {
        sockets: ['*'],
        tags: []
    }
}
```

send WebSocket message to all sockets

```
{
    "object":{
       "id":"541df1c1e5bde554a805b213",
       "createdAt":"2014-09 20T21:29:37.849Z",
       "device":"FZXC0wg0LvaJ",
       "done":false,
       "objectType":"objects.todos",
       "text":"testi2",
       "updatedAt":"2014-09-20T21:29:37.849Z",
       "userId":"user:54196f3f5a3d8b61860381a4"
    },
    "meta":{
        "backendId":"5417e5485a3d8b418a01adb7",
        "eventName":"create",
        "requestId":"a671f67b4f41c20b53988a6057b32539",
        "userId":null}
    }
}
```

Qt Cloud Services

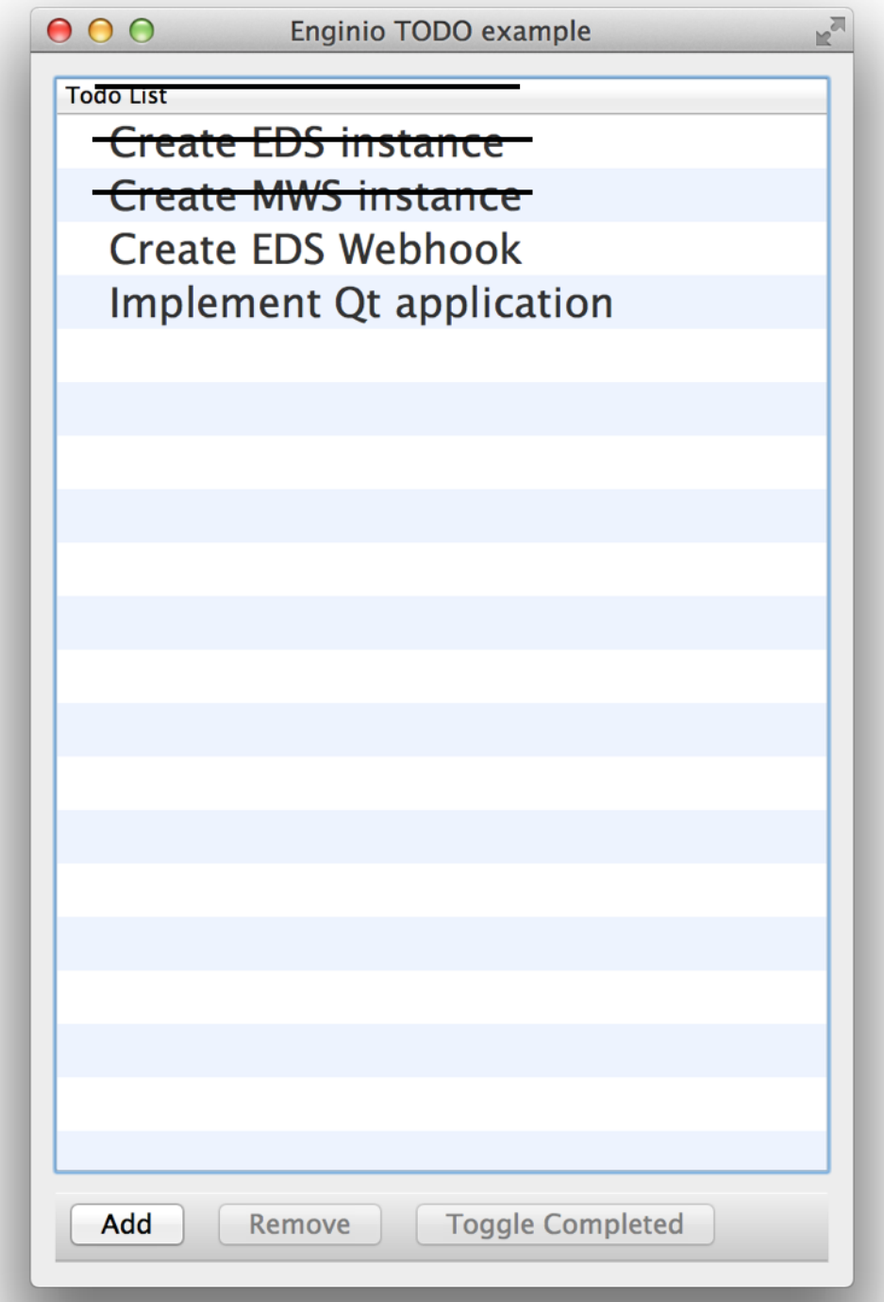
WebSocket server

- Create a EDS instance
- Create a MWS instance
- Create EDS Webhook

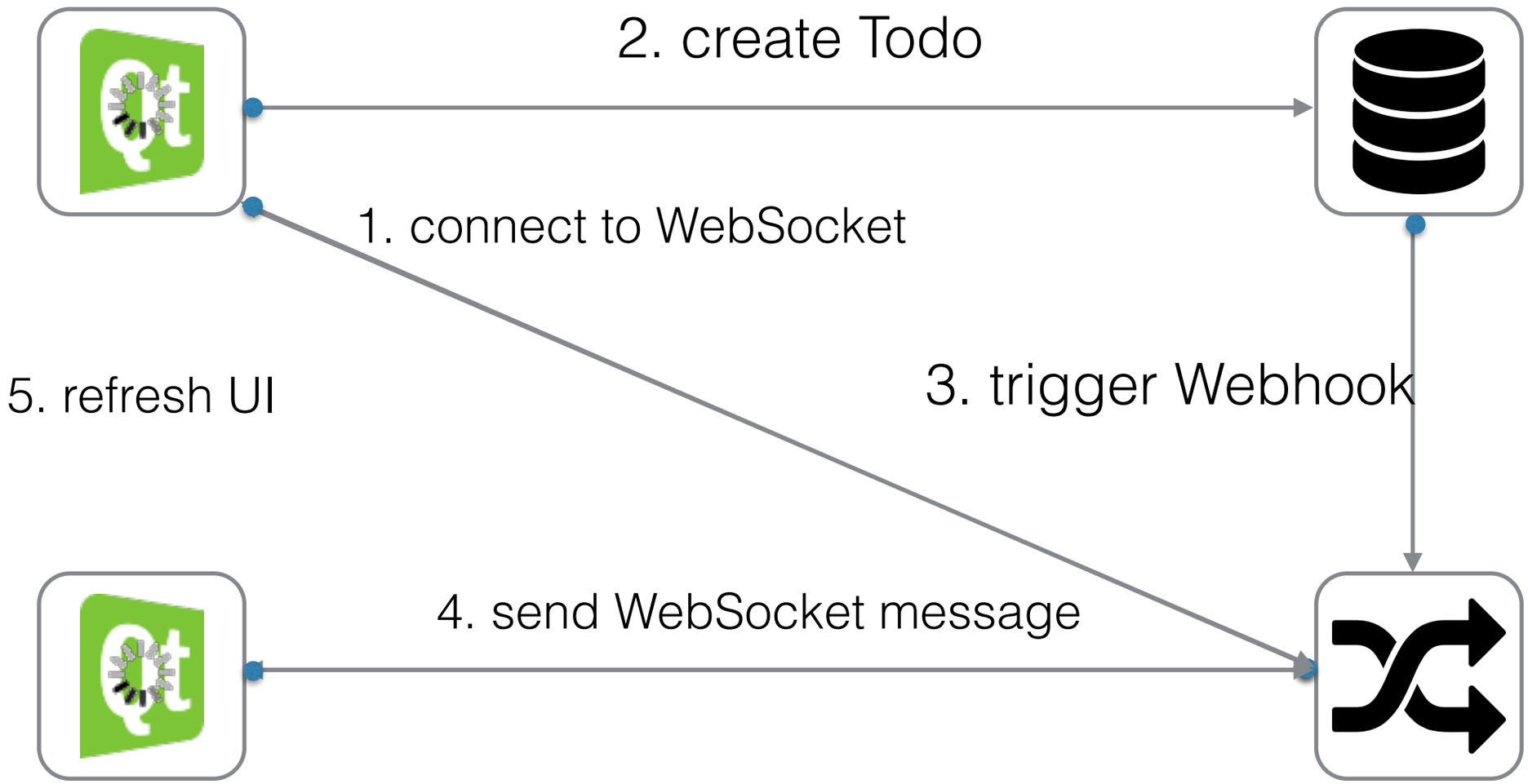
Qt Cloud Services

Enginio Todo

http://bit.ly/qtc-todo



Qt Cloud Services



Qt Cloud Services

websocketclient.cpp

```
#include "websocketclient.h"
#include <QtCore/QDebug>

QT_USE_NAMESPACE

WebSocketClient::WebSocketClient(const QUrl &url, QObject *parent) :
    QObject(parent),
    m_url(url)
{
    connect(&m_webSocket, &QWebSocket::connected, this, &WebSocketClient::onConnected);
    connect(&m_webSocket, &QWebSocket::disconnected, this, &WebSocketClient::closed);
    m_webSocket.open(QUrl(url));
}

void WebSocketClient::onConnected()
{
    qDebug() << "WebSocket connected";
    connect(&m_webSocket, &QWebSocket::textMessageReceived,
            this, &WebSocketClient::onTextMessageReceived);
}

void WebSocketClient::onTextMessageReceived(QString message)
{
    emit onMessageReceived(message);
}
```

Qt Cloud Services

mainwindow.cpp

```
#define REQUEST_URL “https://mws-eu-1.qtc.io/v1/gateways/MWS_GATEWAY_ID/websocket_uri”

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
…
    m_networkManager = new QNetworkAccessManager(this);
    QObject::connect(m_networkManager, &QNetworkAccessManager::finished, this,
&MainWindow::requestFinished);
    getWebSocketUrl();
…
}

QNetworkReply* MainWindow::getWebSocketUrl()
{
    QNetworkRequest request;
    request.setUrl(QUrl(QString("%1").arg(REQUEST_URL)));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

    QNetworkReply *reply = m_networkManager->get(request);
    return reply;
}
void MainWindow::requestFinished(QNetworkReply *reply)
{
    QJsonDocument replyData = QJsonDocument::fromJson(reply->readAll());
    QJsonObject data = replyData.object();

    m_mwsClient = new WebSocketClient(QUrl(data.value("uri").toString()), this);

    QObject::connect(m_mwsClient, &WebSocketClient::onMessageReceived, this,
&MainWindow::messageReceived);
}
```

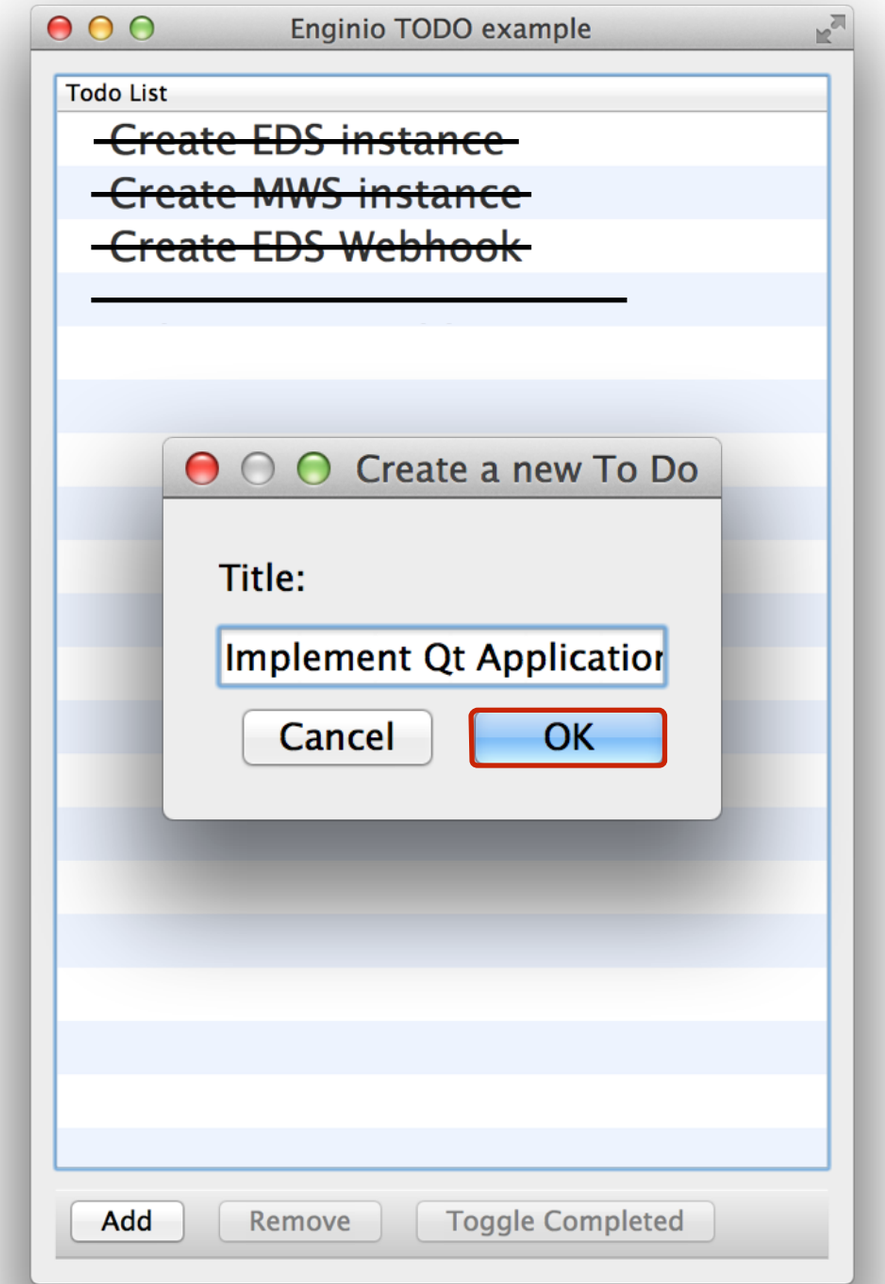
Qt Cloud Services

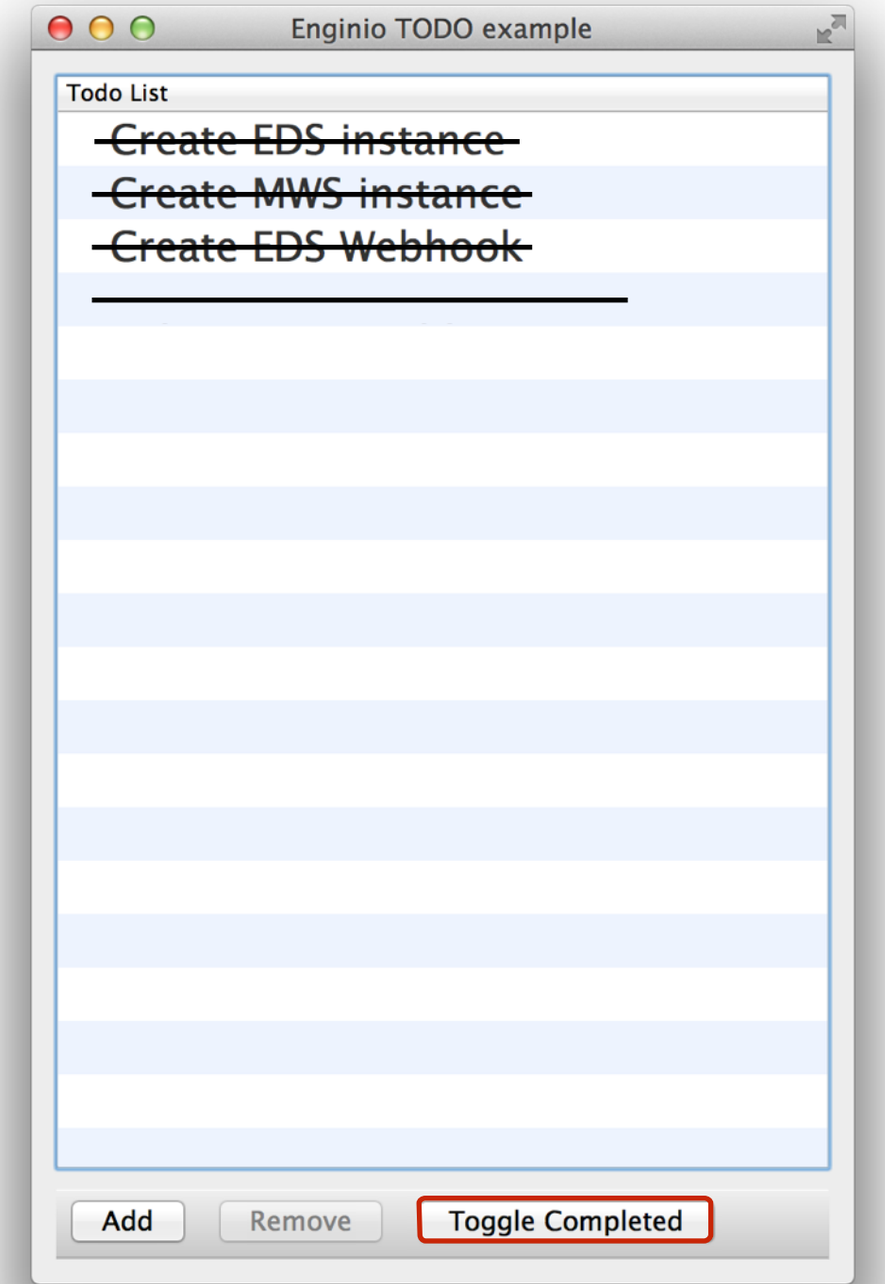
```
void MainWindow::messageReceived(QString message)
{
    qDebug() << "Message received:" << message;
    QJsonDocument messageJson = QJsonDocument::fromJson(message.toUtf8());
    QJsonObject enginioObject = messageJson.object().value("object").toObject();
    QJsonObject metaObject = messageJson.object().value("meta").toObject();
    QString event = metaObject.value("eventName").toString();
    if(event == "delete" || enginioObject.value("device_id").toString() != m_device) {
        qDebug() << "Refresh todos";
        QJsonObject query;
     // reload todos from Enginio
        m_model->setQuery(query);

        query["objectType"] = QString::fromUtf8("objects.todos");
        m_model->setQuery(query);

    }
}
```

Qt Cloud Services

Demo

Qt Cloud Services





Qt Cloud Services

https://flic.kr/p/89crWV

Private sockets

Qt Cloud Services

Qt Cloud Services

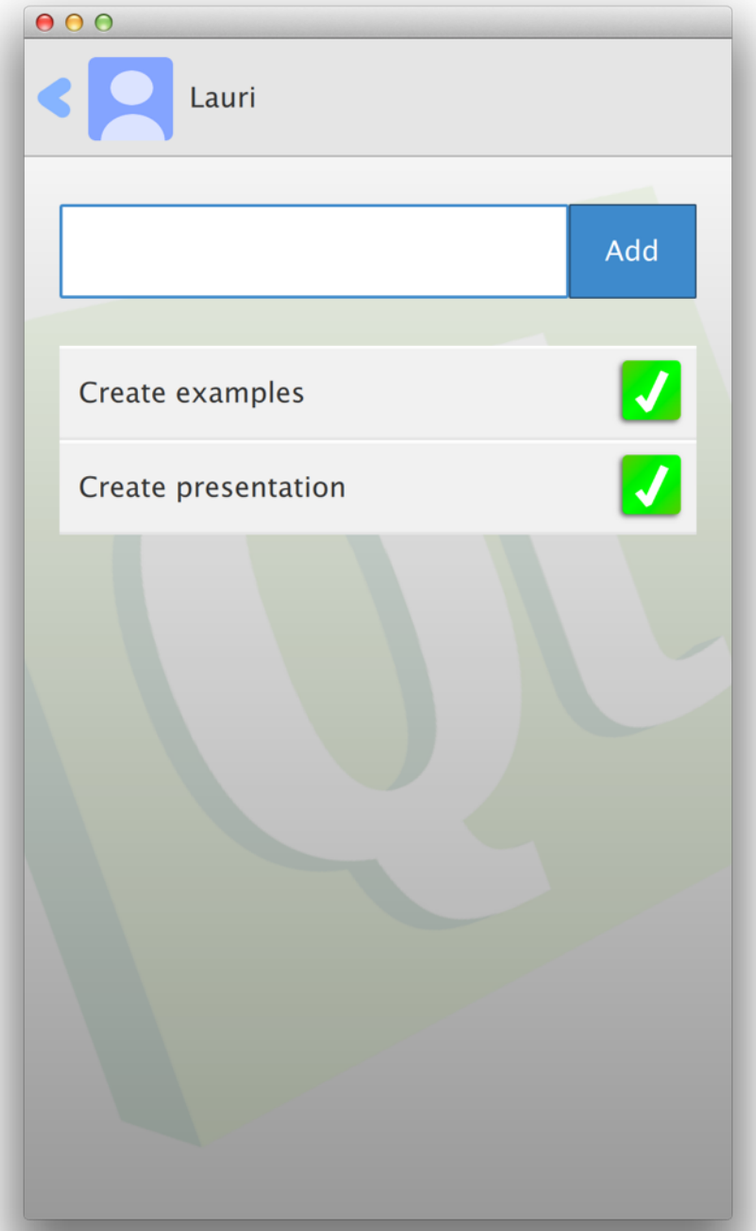
Server-side app

```
// Qt Cloud Services package
var qtc = require('qtc');
var qtcConfig = require('../config/qtc-conf');
var mws = new qtc.Mws(qtcConfig.mws);

module.exports = function(app) {
    app.get('/api/websocket', function(req, res) {
      authenticateRequest(req, function(e, session){
          if(!e){
            mws.createSocket(["user:"+session.userId], function(e, mwsResponse) {
              res.json(mwsResponse);
            });
          } else {
              res.json(error(403, "Access Denied!"));
          }
      });
    });
    app.post('/api/websocket/messages', function(req, res) {
      var payload = req.body.payload;
      console.log('got websocket request:')
      console.log(payload)

      mws.send(JSON.stringify(payload), { sockets: null, tags:
[payload.object.userId] }, function(e, mwsResponse) {
        console.log('send websocket message:');
        console.log(mwsResponse);
        res.json(mwsResponse);
      })
   });
…
}
```

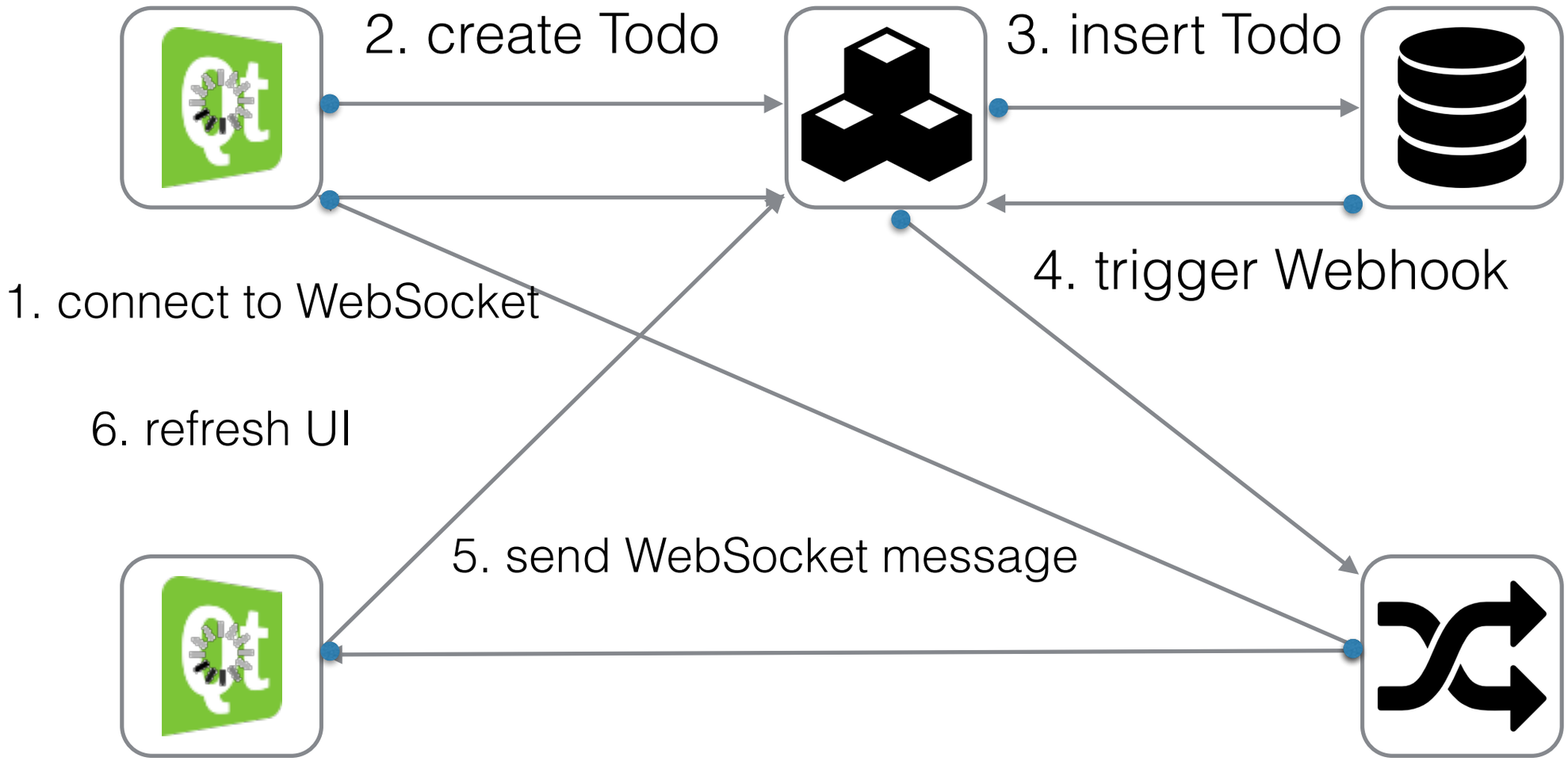
Qt Cloud Services

QML Client

http://bit.ly/qtc-mar-todo



Qt Cloud Services



Qt Cloud Services

main.qml

```
import QtQuick 2.2
import QtQuick.Controls 1.1
import Qt.WebSockets 1.0
import "Storage.js" as Storage

Item {
    id: app
    width: 480
    height: 800
    …

    WebSocket {
        id: socket

        active: true
        onTextMessageReceived: {
            console.log(message)
            Storage.handleWebsocketMessage(message)
        }
        onStatusChanged: {
            if (socket.status == WebSocket.Error) {
              console.log("Error: " + socket.errorString)
            } else if (socket.status == WebSocket.Open) {
              console.log("Open")

            } else if (socket.status == WebSocket.Closed) {
              console.log("Socket closed")
            }
        }
    }
    …
}
```

Qt Cloud Services

storage.js

```
var REQUEST_URL = "http://qtc-tutorial-todo.qtcloudapp.com"

function connectToWebSocket() {
    var doc = new XMLHttpRequest();
    doc.open("GET", REQUEST_URL + "/api/websocket", true);
    doc.setRequestHeader('x-todo-session', sessionId);
    doc.setRequestHeader('Content-Type', 'application/json');
    doc.onreadystatechange = function()
    {
        // When ready
        if (doc.readyState === 4) {
            // If OK
            if (doc.status === 200) {
                var data = JSON.parse(doc.responseText);
                console.log(doc.responseText)
                socket.url = data.uri
            }
        }
    }
    doc.send()
}
```

Qt Cloud Services

```
function handleWebsocketMessage(message) {
    message = JSON.parse(message)
    var event = message.meta.eventName
    var object = message.object
    if(object.device !== device) { // ignore events created by this app
        var i
        if(event === "create") {
            addItem(object.text, object.id)
        }
        else if(event === "update") {
            for (i=itemModel.count; i--;) {
                if( object.id === itemModel.get(i).itemId) {
                    finishItem(i, false)
                    break;
                }
            }
        }
    }
    if(event === "delete") {
        for (i=itemModel.count; i--;) {
            if( object.id === itemModel.get(i).itemId) {
                deleteItem(i, false)
                break;
            }
        }
    }
}
```

Qt Cloud Services

Demo

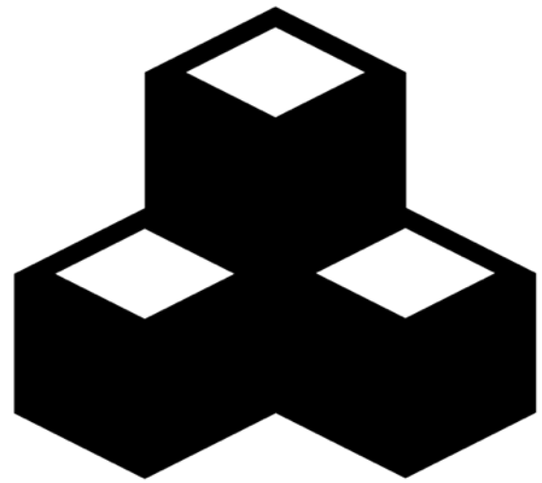
- Try it yourself!
  - http://bit.ly/qtc-todo-demo
  - username: qtincloud
    password: qtincloud

Qt Cloud Services

https://flic.kr/p/cT2gqu

Custom WebSocket Server

Qt Cloud Services



Qt Cloud Services

socket.io

em-websocket

Tornado

… etc …

Qt Cloud Services

I want to use Qt !!!1

Qt Cloud Services

Using WebSockets with MAR

- Server must listen to a port that MAR gives to the app
- To keep idle socket connections alive, the WebSocket server must send a ping control frame to every client at least once in < 60s

Qt Cloud Services

Qt WebSocket EchoServer

http://bit.ly/qt-echoserver

- Example can be found from Qt 5.3 examples
- Bug in Qt 5.3 that prevents deploying it to MAR
- Fixed in Qt 5.4
- Qt 5.4 beta in MAR allows us to test it out

Qt Cloud Services

main.cpp

```
#include <QtCore/QCoreApplication>
#include "echoserver.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int port = 1234;
    if(qEnvironmentVariableIsSet("PORT") && !qEnvironmentVariableIsEmpty("PORT")) {
        port = qgetenv("PORT").toInt();
    }
    EchoServer *server = new EchoServer(port);
    QObject::connect(server, &EchoServer::closed, &a, &QCoreApplication::quit);

    return a.exec();
}
```

Qt Cloud Services

echoserver.cpp

```
EchoServer::EchoServer(quint16 port, QObject *parent) :
    QObject(parent),
    m_pWebSocketServer(new QWebSocketServer(QStringLiteral("Echo Server"),
                                            QWebSocketServer::NonSecureMode, this)),
    m_clients()
{
    if (m_pWebSocketServer->listen(QHostAddress::Any, port)) {
…
        QTimer *timer = new QTimer(this);
        connect(timer, &QTimer::timeout, this, &EchoServer::pingClients);
        timer->start(50000);
    }
}

void EchoServer::pingClients()
{
    if(m_clients.length() > 0) {
        qDebug() << "Ping clients " << m_clients.length();
        for(int i = 0; i< m_clients.length(); i++) {
            m_clients[i]->ping();
        }
    }

}
```

Qt Cloud Services

Demo

Qt Cloud Services

How to Use With Qt?

qtwebsockets-echoserver-example — bash

```
lanevala@it-l-m0015 ~/Development/qtwebsockets-echoserver-example[master*]$ git push qtc master
Counting objects: 1, done.
Writing objects: 100% (1/1), 186 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
-----> Qt app detected
       Installing Qt 5.4.0-beta-2014-09-25_25
-----> Setting up Qt 5.4.0-beta-2014-09-25_25
-----> Configuring with qmake
-----> Compiling with make
       g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIE -DQT_NO_DEBUG -DQT_WEBSOCKETS_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -
I.qtcs/Qt/5.4/gcc_64/mkspecs/linux-g++ -I. -I.qtcs/Qt/5.4/gcc_64/include -I.qtcs/Qt/5.4/gcc_64/include/QtWebSockets -
I.qtcs/Qt/5.4/gcc_64/include/QtNetwork -I.qtcs/Qt/5.4/gcc_64/include/QtCore -I. -o main.o main.cpp
       g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIE -DQT_NO_DEBUG -DQT_WEBSOCKETS_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -
I.qtcs/Qt/5.4/gcc_64/mkspecs/linux-g++ -I. -I.qtcs/Qt/5.4/gcc_64/include -I.qtcs/Qt/5.4/gcc_64/include/QtWebSockets -
I.qtcs/Qt/5.4/gcc_64/include/QtNetwork -I.qtcs/Qt/5.4/gcc_64/include/QtCore -I. -o echoserver.o echoserver.cpp
       /app/.qtcs/Qt/5.4/gcc_64/bin/moc -DQT_NO_DEBUG -DQT_WEBSOCKETS_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -I/app/.qtcs/Qt/
5.4/gcc_64/mkspecs/linux-g++ -I/tmp/build -I/app/.qtcs/Qt/5.4/gcc_64/include -I/app/.qtcs/Qt/5.4/gcc_64/include/
QtWebSockets -I/app/.qtcs/Qt/5.4/gcc_64/include/QtNetwork -I/app/.qtcs/Qt/5.4/gcc_64/include/QtCore echoserver.h -o
moc_echoserver.cpp
       g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIE -DQT_NO_DEBUG -DQT_WEBSOCKETS_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -
I.qtcs/Qt/5.4/gcc_64/mkspecs/linux-g++ -I. -I.qtcs/Qt/5.4/gcc_64/include -I.qtcs/Qt/5.4/gcc_64/include/QtWebSockets -
I.qtcs/Qt/5.4/gcc_64/include/QtNetwork -I.qtcs/Qt/5.4/gcc_64/include/QtCore -I. -o moc_echoserver.o moc_echoserver.cpp
       g++ -Wl,-O1 -Wl,-rpath,/app/.qtcs/Qt/5.4/gcc_64 -Wl,-rpath,/app/.qtcs/Qt/5.4/gcc_64/lib -o echoserver main.o
echoserver.o moc_echoserver.o   -L/app/.qtcs/Qt/5.4/gcc_64/lib -lQt5WebSockets -lQt5Network -lQt5Core -lpthread
-----> Discovering process types
       Procfile declares types -> web
-----> Compiled slug size is 108M
-----> Deploying app
       Uploading app container ...... done.
       mar-eu-1-twwto7g0 deployed to http://mar-eu-1-twwto8g1.qtcloudapp.com
```

Qt Cloud Services

WebSocket Echo Client

Connect | Disconnect | State

```
CONNECTED
Does it echo?
```

Does it echo? | Send

```
qtwebsockets-echoserver-example — bash
Echoserver listening on port 5000
Echoserver got new connection
Ping clients  1
Echoserver got new pong
Echoserver got new message "Does it echo?"
```

Qt Cloud Services

Qt WebChannel

- The Qt WebChannel module provides a library for seamless integration of C++ and QML applications with HTML/JavaScript clients.

- Any QObject can be published to remote clients, where its public API becomes available.

Qt Cloud Services

Example

http://bit.ly/qtc-webchannel

- Qt WebChannel Standalone example from Qt 5.4 examples

- Instead of local WebSocket server, we are using WebSocket server running on MAR

Qt Cloud Services

WebSocketClientWrapper.cpp

```
/*!
    Construct the client wrapper with the given url.

    All clients connecting to the QWebSocketServer will be automatically wrapped
    in WebSocketTransport objects.
*/
WebSocketClientWrapper::WebSocketClientWrapper(const QUrl &url, QObject *parent)
    : QObject(parent)
    , m_url(url)
{
    qWarning() << "Open WebSocket connection: " << m_url;
    connect(&m_webSocket, &QWebSocket::connected,
            this, &WebSocketClientWrapper::handleNewConnection);
    m_webSocket.open(url);
}

/*!
    Wrap an incoming WebSocket connection in a WebSocketTransport object.
*/
void WebSocketClientWrapper::handleNewConnection()
{
    qWarning() << "WebSocket connected: " << m_url;
    emit clientConnected(new WebSocketTransport(&m_webSocket));
}
```

Qt Cloud Services

main.cpp

```
// wrap WebSocket clients in QWebChannelAbstractTransport objects
WebSocketClientWrapper clientWrapper(QUrl("wss://mar-eu-1-twwto7g0.qtcloudapp.com"));

// setup the channel
QWebChannel channel;
QObject::connect(&clientWrapper, &WebSocketClientWrapper::clientConnected,
                 &channel, &QWebChannel::connectTo);

// setup the dialog and publish it to the QWebChannel
Dialog dialog;
channel.registerObject(QStringLiteral("dialog"), &dialog);
```

Qt Cloud Services

index.html

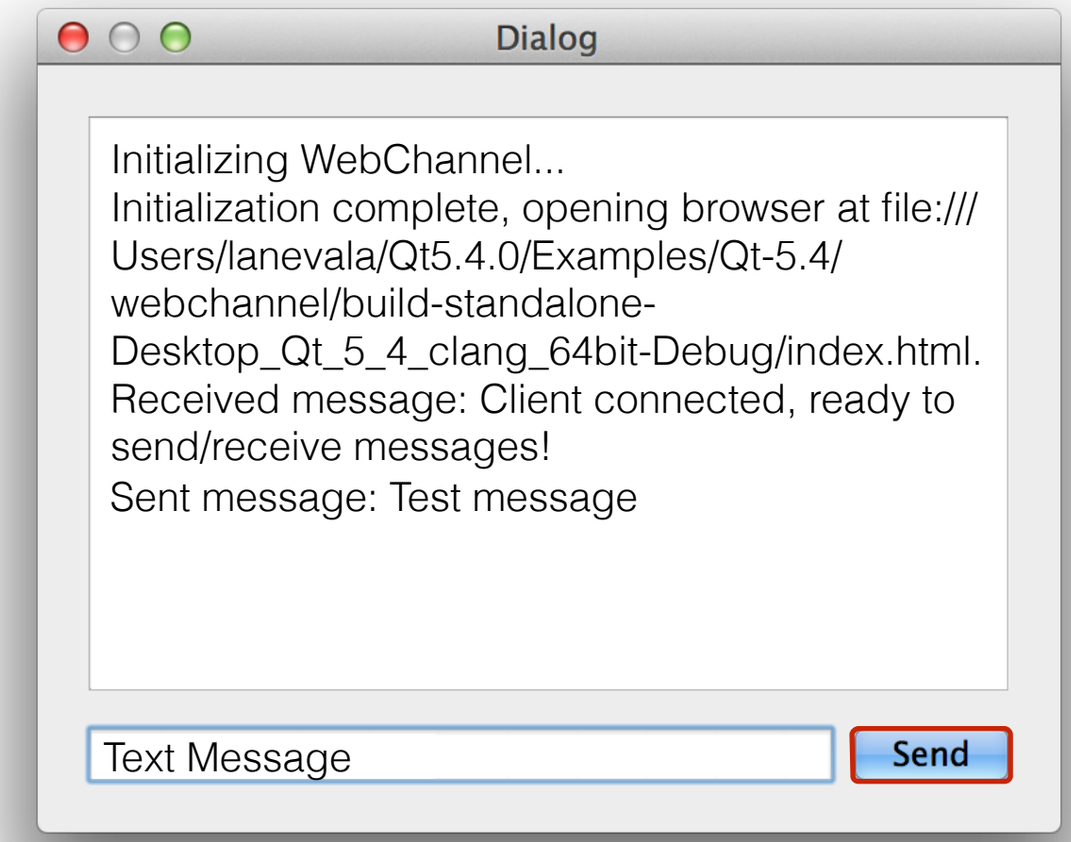
```
window.onload = function() {
    var baseUrl = "wss://mar-eu-1-twwto7g0.qtcloudapp.com";
    output("Connecting to WebSocket server at " + baseUrl + ".");
    var socket = new WebSocket(baseUrl);
    socket.onopen = function()
    {
        output("WebSocket connected, setting up QWebChannel.");
        new QWebChannel(socket, function(channel) {
            // make dialog object accessible globally
            window.dialog = channel.objects.dialog;

            document.getElementById("send").onclick = function() {
                var input = document.getElementById("input");
                var text = input.value;
                if (!text) {
                    return;
                }

                output("Sent message: " + text);
                input.value = "";
                dialog.receiveText(text);
            }

            dialog.sendText.connect(function(message) {
                output("Received message: " + message);
            });

            dialog.receiveText("Client connected, ready to send/receive messages!");
            output("Connected to WebChannel, ready to send/receive messages!");
        });
    }
}
```

Qt Cloud Services

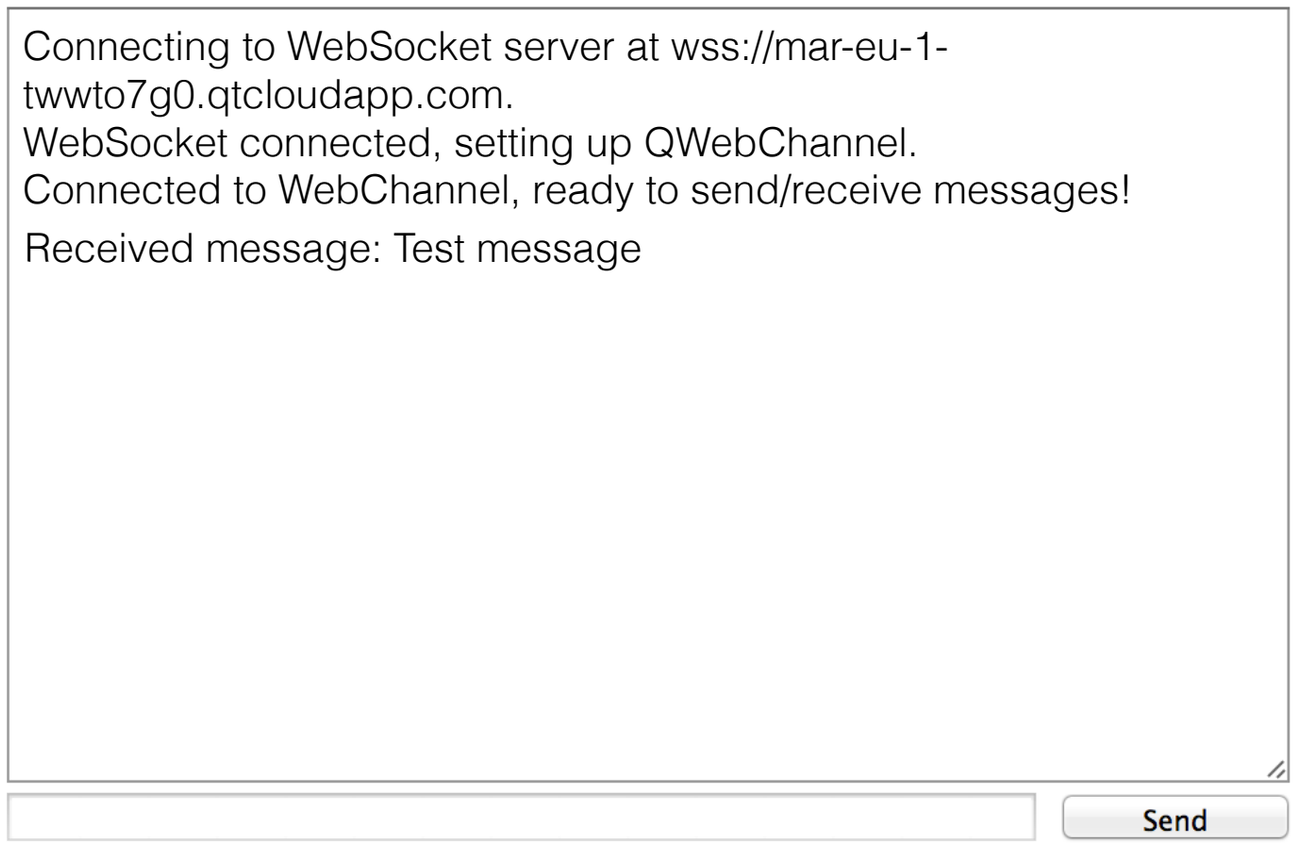
Demo

Qt Cloud Services

QML



HTML



Qt Cloud Services

Summary

- WebSockets are great
- Using WebSockets with Qt and Qt Cloud Services is super easy and fast
- There are multiple levels on you can utilise Qt Cloud Services to send and receive WebSocket messages
  - Use Managed WebSocket service
  - Deploy your own solution to MAR

Qt Cloud Services

Further actions

- Visit qtcloudservices.com
- Sign-up
- Use it
- Give us feedback
  - Qt forum
  - info@qtcloudservices.com

Qt Cloud Services

Thank you!

Source Codes:

- Enginio Todo:
  http://bit.ly/qtc-todo

- Qt Cloud Services Todo:
  http://bit.ly/qtc-mar-todo

- Qt WebSocket Echo Server:
  http://bit.ly/qt-echoserver

Qt Cloud Services