Яндекс

Яндекс

Разработка приложения под Cocaine — быстро, просто, удобно

Александр Пономарев

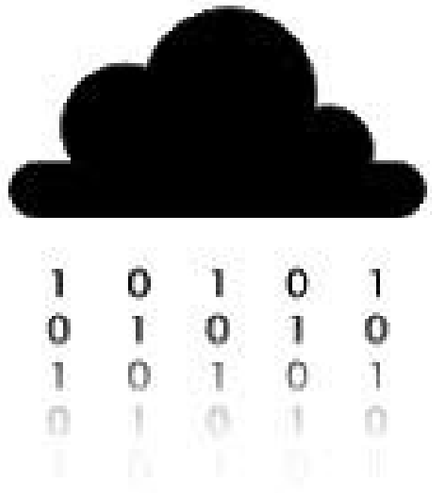
Разработчик

Перевод в облаке

Сделай настолько просто, насколько это возможно, но не проще.

Работник патентного бюро

Перевод в облаке



- Возьмем открытое api переводов, например translate.yandex.ru
- Выберем транспорт — пусть будет jabber (gloox)
- Поместим все это в облако

Попробуйте сами

Попробуйте облако сами:

jabber: cloud-test@jabber.ru

Пример: tr облако переводи текст

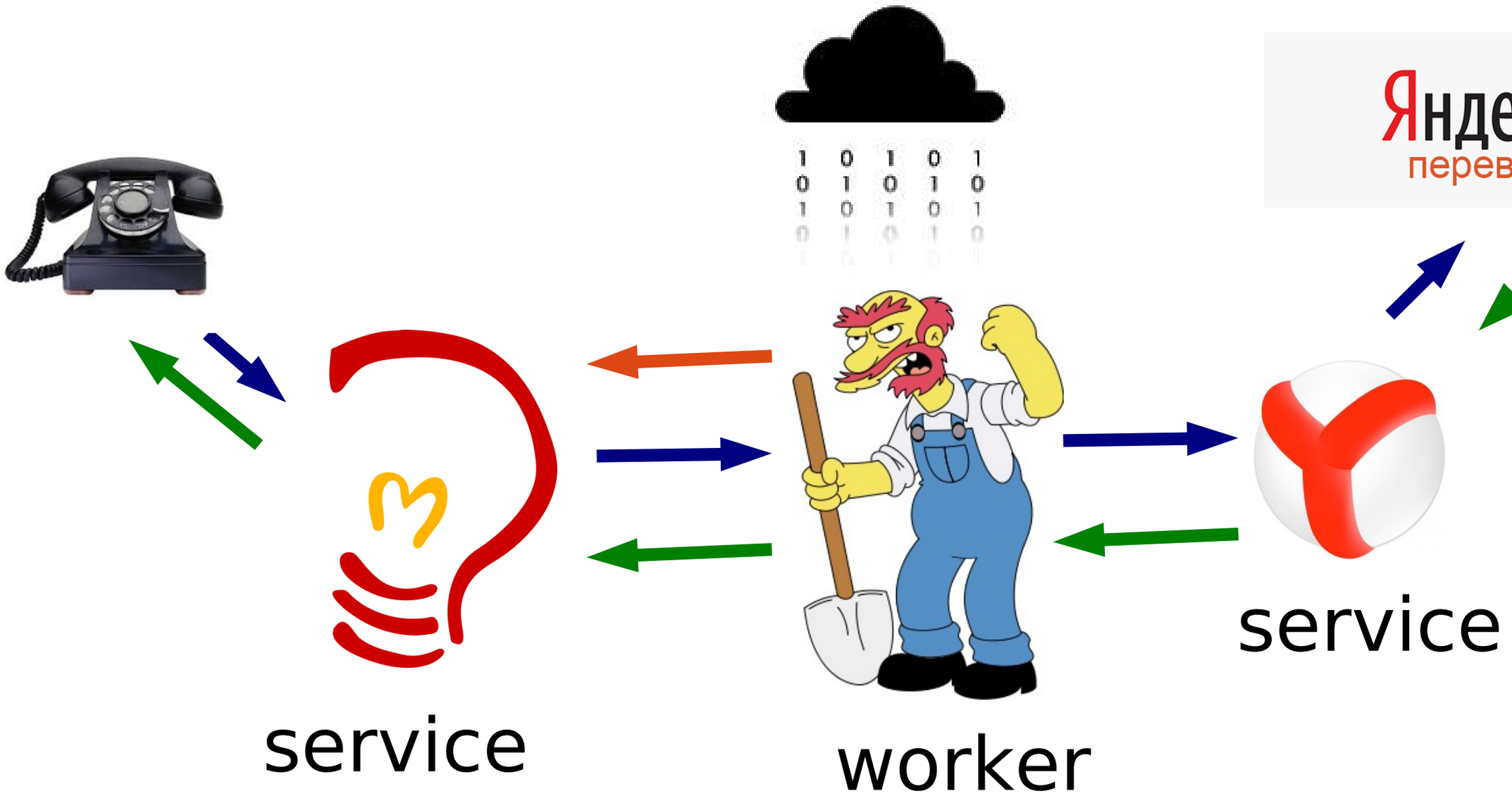
Пример: tr облако копай огород

Пример: tr cloud is your best friend

Пример: tr Cloud dient das große Ziel

Префикс: tr_пробел_ваш текст

Как все будет работать?



Инициализируем приложение

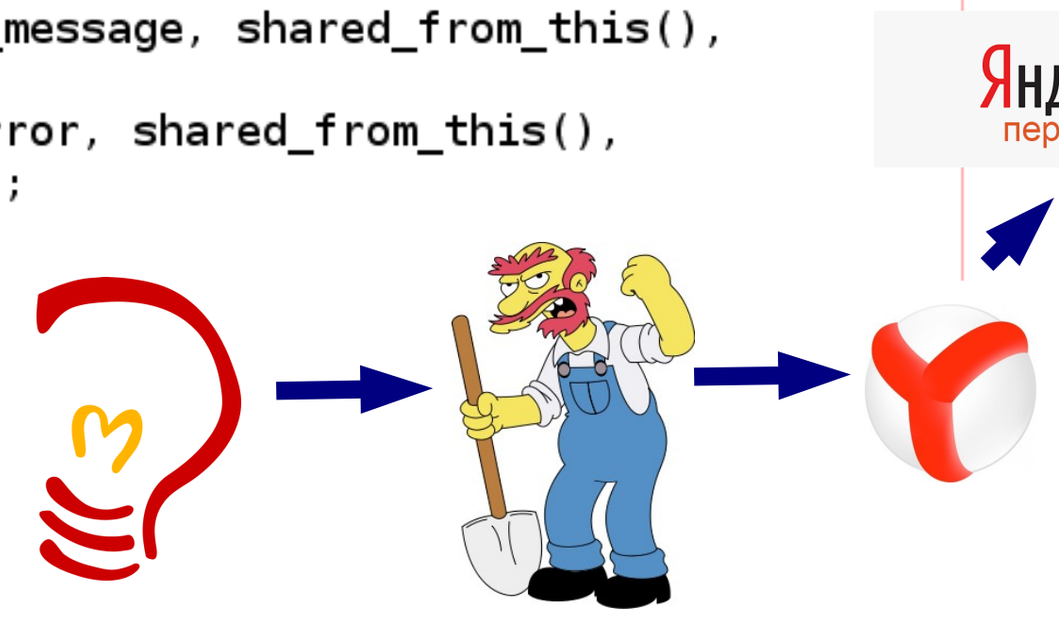
Достаем из облака сервисы:

```
class AppProxy :
    public application<AppProxy>

int
main(int argc, char *argv[])
{
    cocaine::framework::worker_t::run<AppProxy>(argc, argv);
}

void
initialize() {
    log_ = service_manager()->get_system_logger();

    jabber_ = service_manager()->
            get_service<cocaine::framework::jabberer_service_t>("jabber");
    urlfetch_ = service_manager()->
            get_service<cocaine::framework::urlfetcher_service_t>("urlfetch");

    jabber_->get_messages(MESSAGE_PREFIX)
        .on_message(bind(&AppProxy::on_message, shared_from_this(),
                std::placeholders::_1))
        .on_error(bind(&AppProxy::on_error, shared_from_this(),
                std::placeholders::_2));
}
```

Обрабатываем ответы

Получаем сообщения из сервиса и определяем язык:

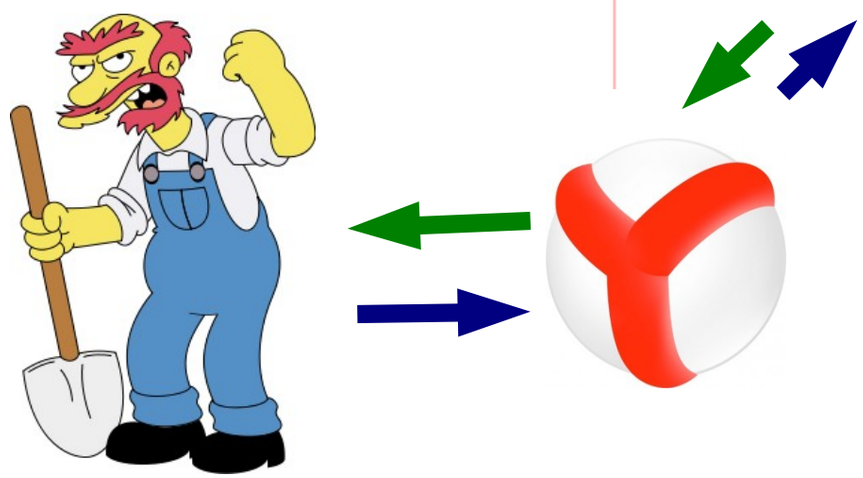
```
void
on_message(const vector<pair<string, string>>& messages) {
    BOOST_FOREACH(auto& message, messages) {
        string url = boost::str( boost::format(
            "https://translate.yandex.net/api/v1.5/tr.json/detect?key=%1%&text=%2%")
            % TRANSLATE_KEY % urlencode(message.second) );

        COCAINE_LOG_INFO(log_, "detect url %s", url);

        urlfetch_->get(url, 5000)
            .on_message(bind(&AppProxy::on_detect, shared_from_this(), message,
                std::placeholders::_1, std::placeholders::_2))
            .on_error(bind(&AppProxy::on_error, shared_from_this(),
                std::placeholders::_2));
    }

    jabber_->get_messages(MESSAGE_PREFIX)
        .on_message(bind(&AppProxy::on_message, shared_from_this(),
                std::placeholders::_1))
        .on_error(bind(&AppProxy::on_error, shared_from_this(),
                std::placeholders::_2));
}
```



Обрабатываем ответы

```
void on_detect(const pair<string, string>& message, bool success,
    const string& response_json) {
    if (!success) return;

    json_spirit::mValue response;
    json_spirit::read_string(response_json, response);
    string lang = get_from_json<string>(response, "lang");

    if (lang == "ru") lang = "ru-en";
    else lang = boost::str( boost::format("%1%-%2%") % lang % "ru" );

    string url = boost::str( boost::format(
                "https://translate.yandex.net/api/v1.5/tr.json/translate?key=%1%&lang=
                % TRANSLATE_KEY % lang % urlencode(message.second) );

    COCAINE_LOG_INFO(log_, "Language detected: %s", lang);
    COCAINE_LOG_INFO(log_, "translate url %s", url);

    urlfetch_->get(url, 5000)
            .on_message(bind(&AppProxy::on_translate, shared_from_this(),
                message, std::placeholders::_1, std::placeholders::_2))
            .on_error(bind(&AppProxy::on_error, shared_from_this(),
                std::placeholders::_2));
}
```

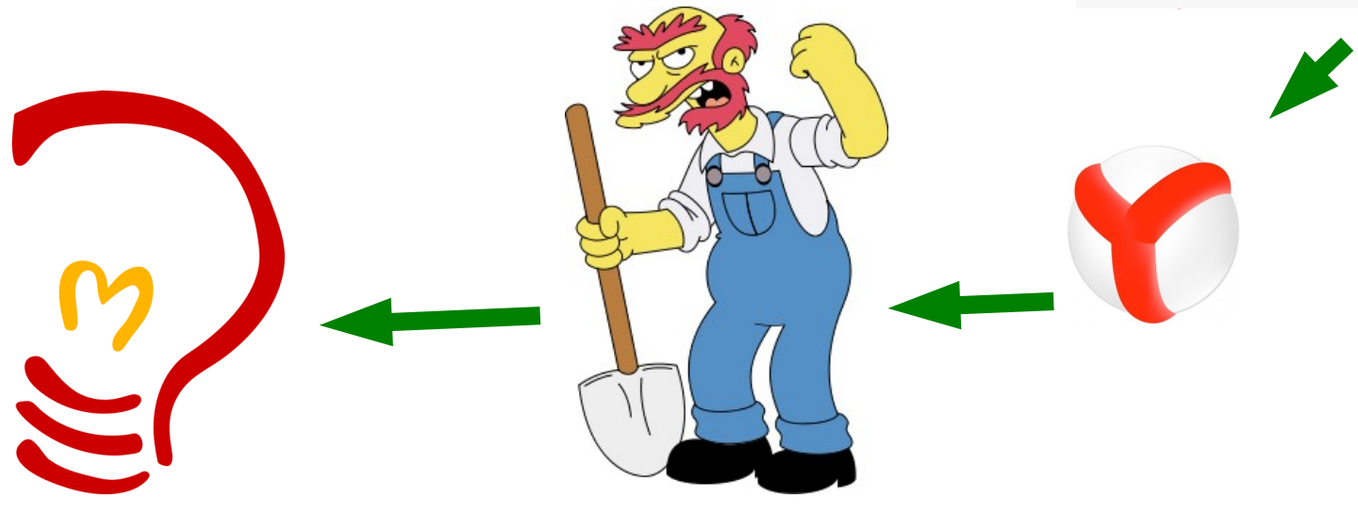
Яндекс
перевод



Обрабатываем ответы

Получив перевод посылаем его в джаббер:

```
void on_translate(const pair<string, string>& message, bool success,
    const string& response_json) {
    if (!success) return;

    json_spirit::mValue response;
    json_spirit::read_string(response_json, response);

    const json_spirit::mArray& texts =
        get_from_json<json_spirit::mArray>(response, "text");

    std::stringstream ss;
    BOOST_FOREACH(const json_spirit::mValue& text, texts) {
        ss << text.get_str() << " ";
    }

    jabber_->send_message(message.first, ss.str());
}
```



Сервис

Составим протокол

```
namespace jabber {
    struct get_messages {
        typedef jabber_tag tag;

        typedef boost::mpl::list<
            /* tag */ std::string
        > tuple_type;

        typedef boost::mpl::list<
            /* messages */ std::vector<std::pair<std::string, std::string>>
        > result_type;
    };

    struct send_message {
        typedef jabber_tag tag;

        typedef boost::mpl::list<
            /* jabber_id */ std::string,
            /* message */ std::string,
            /* subject */ optional<std::string>
        > tuple_type;

        typedef void result_type;
    };
}
```

Составим протокол

```
template<>
struct protocol<jabber_tag> {
    typedef mpl::list<
        jabber::get_messages,
        jabber::send_message
    > type;

    typedef boost::mpl::int_<
        1
    >::type version;
};
```

Объявим класс реализующий протокол

```
class jabber_t:
    public api::service_t,
    public gloox::MessageHandler,
    public RosterListener
{
    public:
        jabber_t(context_t& context,
                 io::reactor_t& reactor,
                 const std::string& name,
                 const Json::Value& args);

        ~jabber_t();

        void
        start_jabber(const std::string& login, const std::string& password);

        virtual void
        handleMessage( const gloox::Message& incoming_msg,
            gloox::MessageSession* session = 0);

    private:
        deferred<get_tuple_t>
        get_messages(const std::string& prefix);

        void
        send_message(const jabber_id_t& jabber_id,
             const std::string& message,
             const std::string& subject);
```

Добавим stub для c++

```
class jabberer_service_t:
    public service_t
{
    public:
        jabberer_service_t(const std::string& name,
                           cocaine::io::reactor_t& service,
                           const cocaine::io::tcp::endpoint& resolver,
                           std::shared_ptr<logger_t> logger) :
            service_t(name, service, resolver, logger, version_type())
        { }

        service_t::handler<io::jabber::get_messages>::future
        get_messages(const std::string& tag) {
            return call<io::jabber::get_messages>(tag);
        }

        service_t::handler<io::jabber::send_message>::future
        send_message(const std::string& jabber_id,
                     const std::string& message) {
            return call<io::jabber::send_message>(jabber_id, message);
        }

        service_t::handler<io::jabber::send_message>::future
        send_message(const std::string& jabber_id,
                     const std::string& message,
                     const std::string& subject) {
            return call<io::jabber::send_message>(jabber_id, message, subject);
        }
};
```

Инициализация сервиса

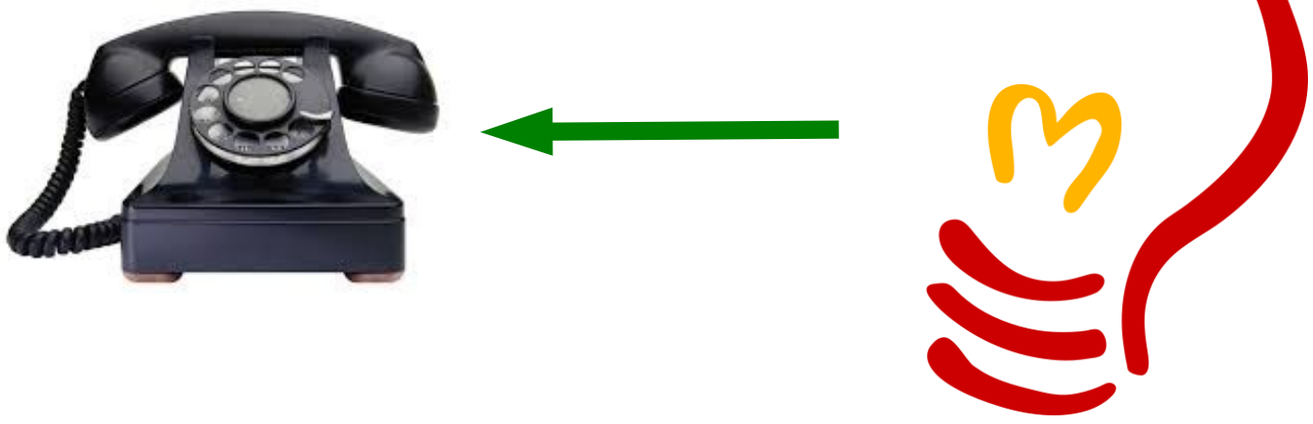
```
jabber_t::jabber_t(context_t& context,
                   reactor_t& reactor,
                   const std::string& name,
                   const Json::Value& args):
    service_t(context, reactor, name, args),
    log_(new logging::log_t(context, name)),
    alive_(true)
{
    on<io::jabber::get_messages>("get_messages",
            std::bind(&jabber_t::get_messages, this, ph::_1));
    on<io::jabber::send_message>("send_message",
            std::bind(&jabber_t::send_message, this, ph::_1, ph::_2, ph::_3));

    std::string login = args["login"].asCString();
    std::string password = args["password"].asCString();

    for (auto it = args["prefixes"].begin(); it != args["prefixes"].end(); ++it)
        allowed_prefixes_.insert((*it).asCString());
    }

    boost::thread(boost::bind(&jabber_t::start_jabber, this, login, password));
}
```

Запуск джаббера

```
void
jabber_t::start_jabber(const std::string& login, const std::string& password) {
    COCAINE_LOG_INFO(log_, "Starting jabber client" );

    while (alive_) {
        try {
            JID jid( login );
            client_.reset(new Client(jid, password));
            client_->registerMessageHandler( this );
            client_->rosterManager()->registerRosterListener( this );
            client_->connect();
        } catch (std::exception& e) {
            COCAINE_LOG_ERROR(log_,
                "Error occured while starting or running jabber %s", e.what() );
        }

        sleep(10);
    }

    COCAINE_LOG_INFO(log_, "Jabber client has stopped" );
}
```

Реализация метода

```
void
jabber_t::send_message(const jabber_id_t& jabber_id,
                const std::string& message,
                const std::string& subject)
{
    JID jid( jabber_id );
    Message msg( Message::Normal, jid, message, subject );
    client_->send(msg);
    COCAINE_LOG_INFO(log_, "Message sent to %s", jabber_id);
}
```

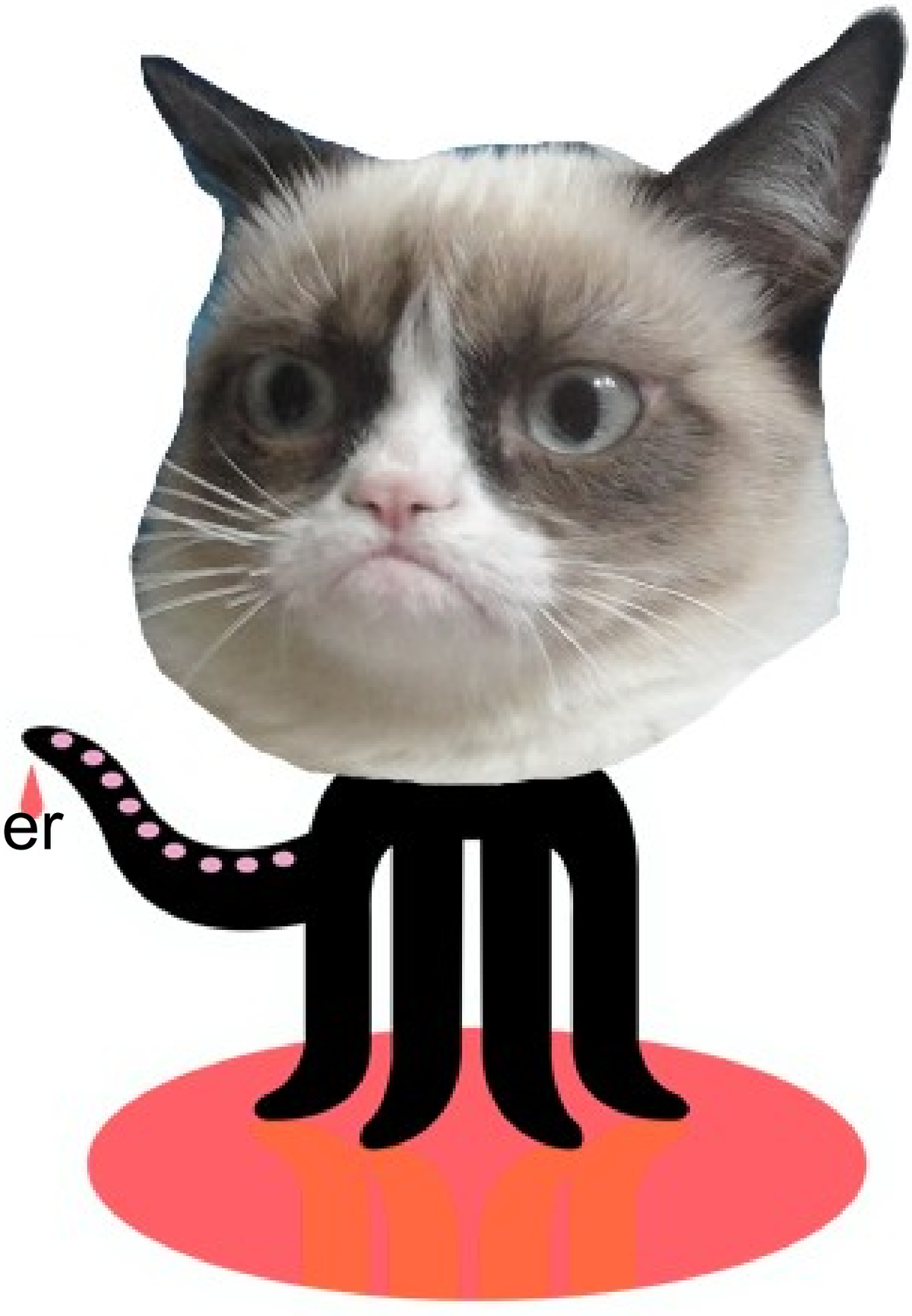


Где взять

Облачная платформа:
bit.ly/iwantcoke

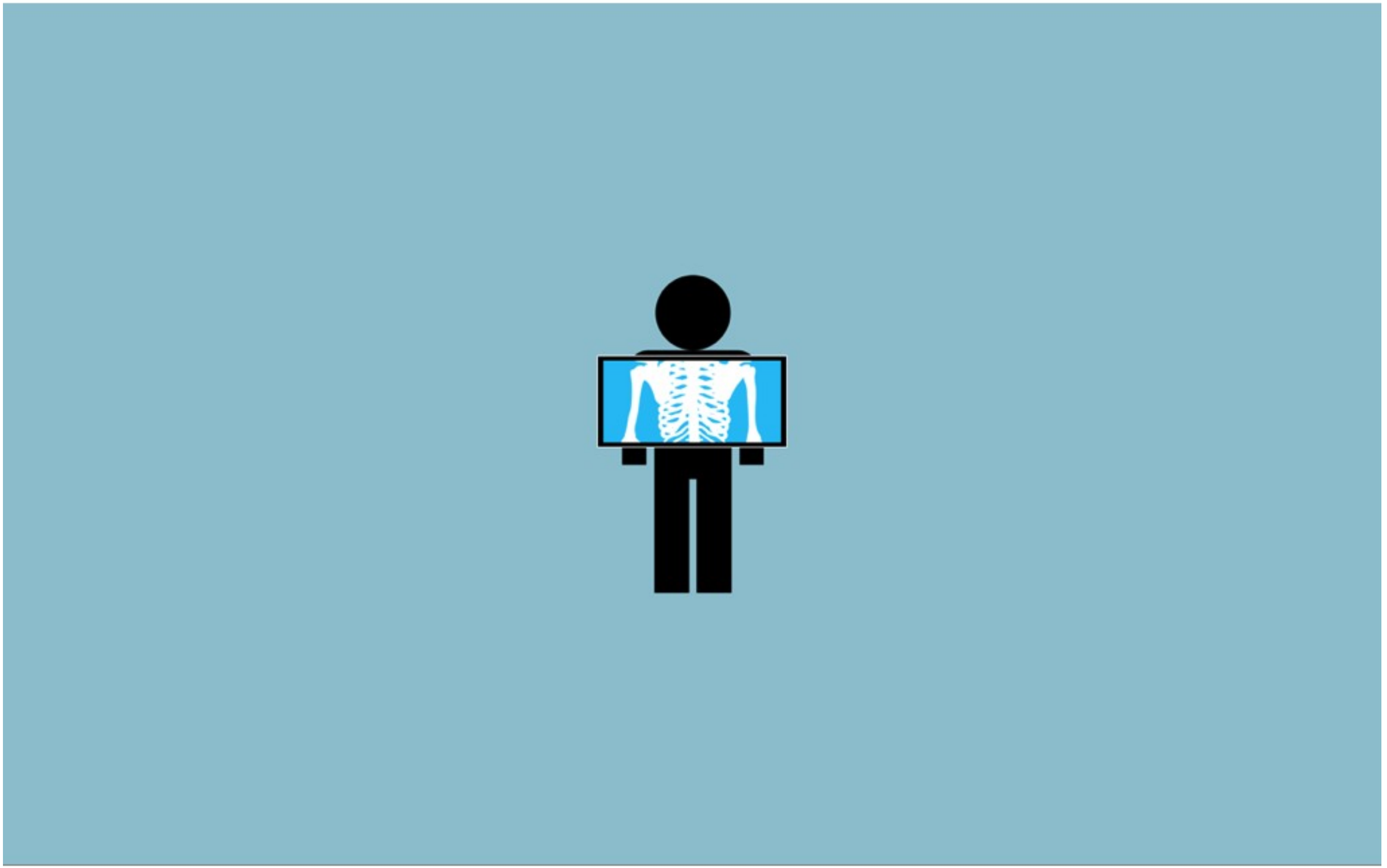
Сервис jabber:
bit.ly/cocaine-service-jabber

Приложение:
bit.ly/translation-cloud



Python

```
from cocaine.servicres import Service
from cocaine.worker import Worker

TRANSLATE_KEY = "<some_key>" # key for yandex api

jabber = Service("jabber")
urlfetch = Service("urlfetch")

"https://translate.yandex.net/api/v1.5/tr.json/translate?key=%1%&lang=%2%&text=%3%"

def translate(request, response):
    messages = yield jabber.get_messages("tr")
    for from, what in messages:
        url = "https://translate.yandex.net/api/v1.5/tr.json/detect?key=%s&text=%s"
             % (TRANSLATE_KEY, message)
        api_resp = yield urlfetch.get(url, 5000)
        lang = api_resp["lang"]
        if (lang == "ru"):
            lang = "ru-en"
        else:
            lang = "%s-%s" % (lang, "ru")
        url = "https://translate.yandex.net/api/v1.5/tr.json/translate?key=%s&lang=
             %s&text=%s" % (TRANSLATE_KEY, lang, message)
        trans_resp = yield urlfetch.get(url, 5000)
        yield jabber.send_message(from, " ".join(trans_resp['text']))
    response.write("DONE")
    response.close()
```



Взаимодействие Worker - Cocaine-Runtime

Жизнь worker'а после старта

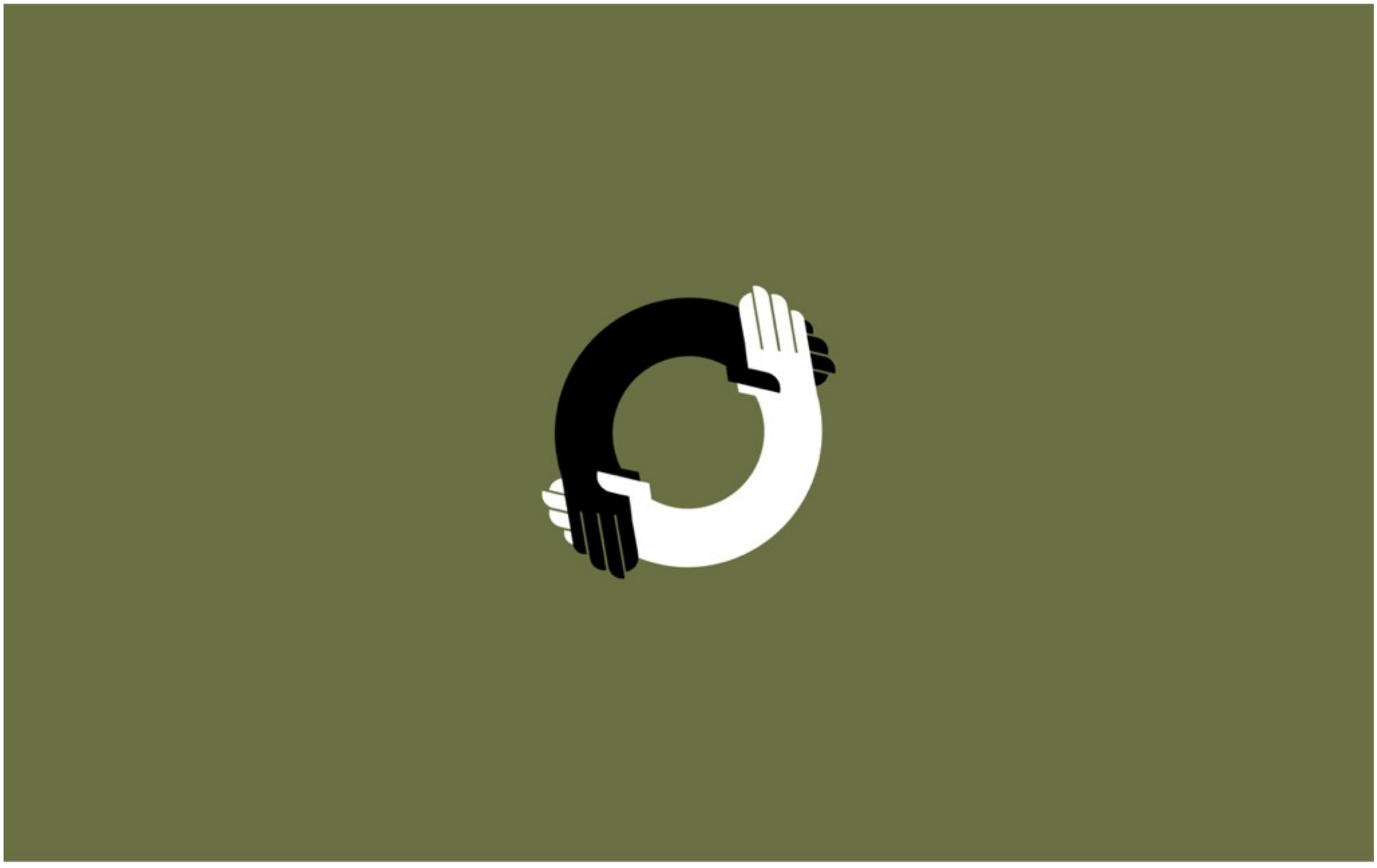
Инициализироваться

- Подключиться на сокет
- Послать **handshake**
- По окончании послать **heartbeat**

Жизнь worker'а после старта

Быть готовым:

- Начать сессию на **invoke**
- Доставить **chunk** в обработчик события
- Транслировать **chunk**'и с ответами в cocaine-runtime
- Закрыть сессию на **choke**
- Периодически слать **heartbeat** cocaine-runtime'у



Взаимодействие Worker - Service

Worker — Service: протокол един

Инициализировать:

- Сходить в сервис локатор за **endpoint**'ом
- В динамических языках — создать “на лету” интерфейс к сервису
- Законнектиться на сокет сервиса

Worker — Service: теперь мы раздаем задания

Просто работать:

- Слать из обработчика задания в сервис (**invoke**, **chunk**, **choke**)
- Перенаправлять ответы в обработчик

Яндекс

Александр Пономарев

Разработчик

noname@yandex-team.ru

Спасибо