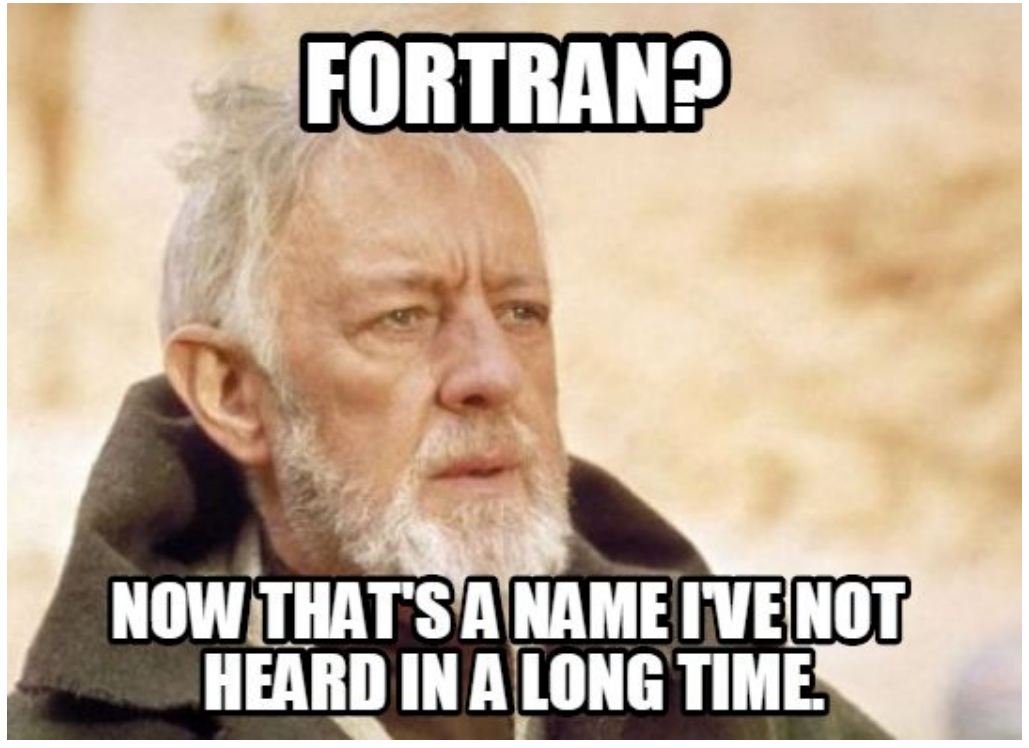
Connecting Fortran to the Internet of Things

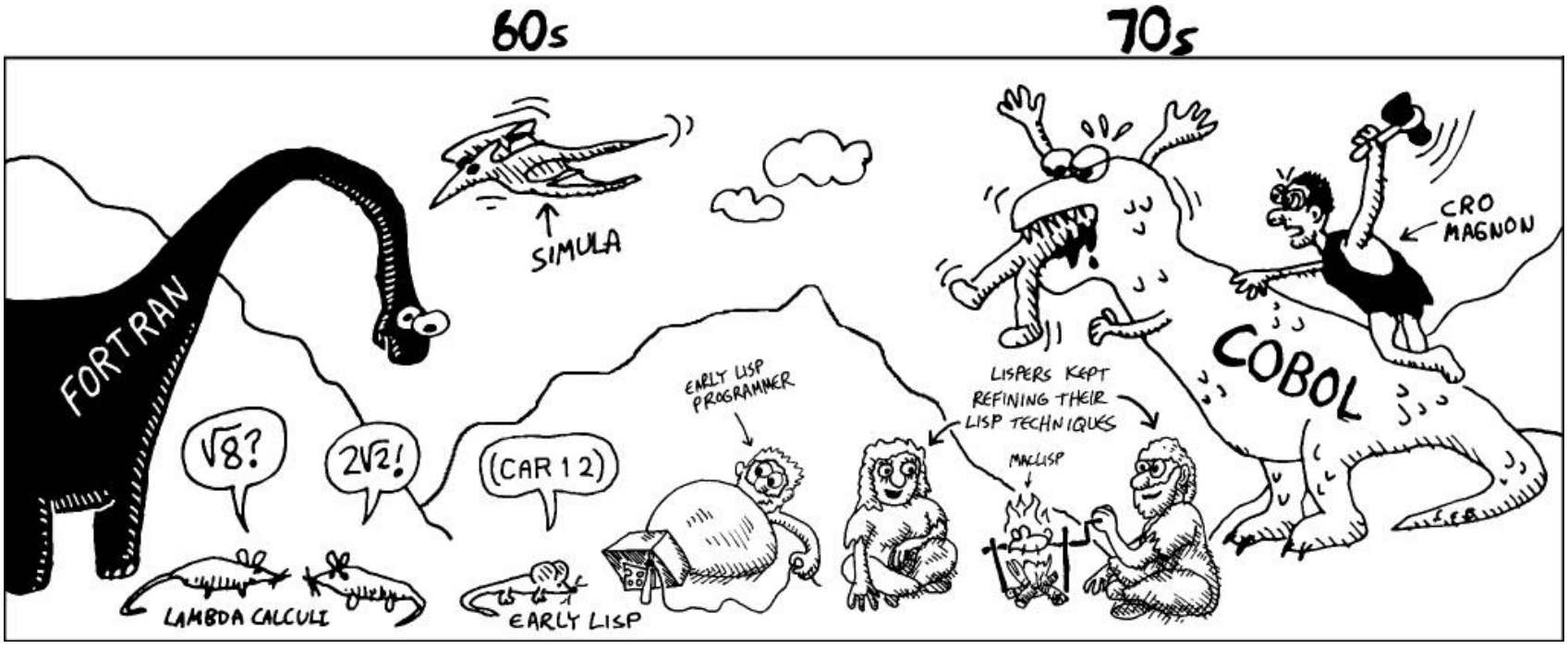
Philipp Engel

Hochschule Neubrandenburg – University of Applied Sciences
Faculty of Landscape Sciences and Geomatics
Germany

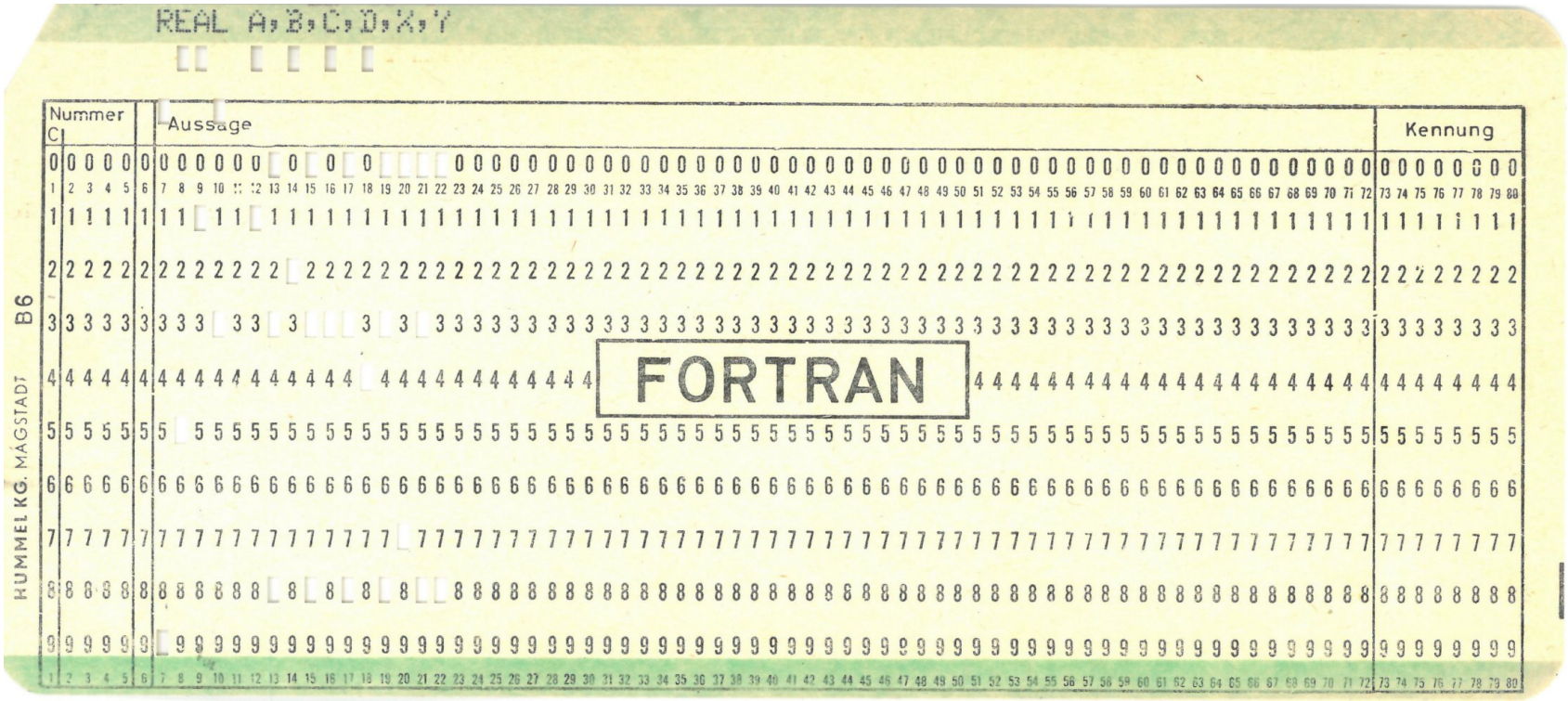
3 July 2020

Bundesministerium
für Verkehr und
digitale Infrastruktur


FORTRAN?
NOW THAT'S A NAME I'VE NOT HEARD IN A LONG TIME.

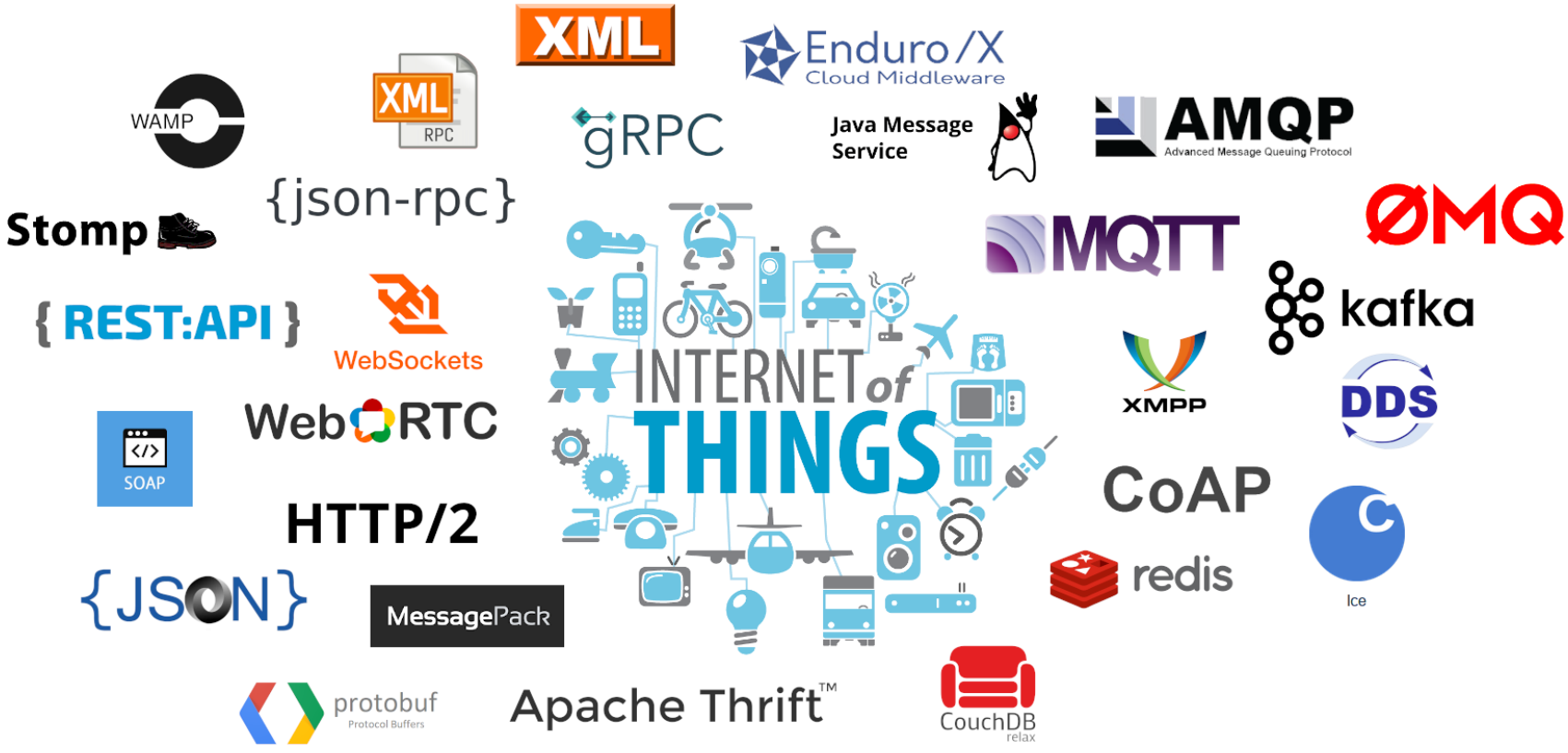


Source: Land of Lisp



Source: Wikimedia Commons

Source: IBM


XML
Enduro/X
Cloud Middleware
WAMP
XML
RPC
gRPC
Java Message
Service
AMQP
Advanced Message Queuing Protocol
{json-rpc}
Stomp
MQTT
ØMQ
{ REST:API }
WebSockets
kafka
INTERNET of
THINGS
XMPP
DDS
SOAP
WebRTC
CoAP
HTTP/2
redis
Ice
{JSON}
MessagePack
protobuf
Protocol Buffers
Apache Thrift™
CouchDB
relax

Internet of Things: Vision

- inter-connected “smart” devices

Internet of Things: Vision

- inter-connected “smart” devices
  - industrial automation
  - wireless sensor networks
  - vehicular communication
  - monitoring and alerting
  - agriculture
  - manufacturing
  - home automation
  - entertainment
  - telemedicine
  - transportation
  - web services

Internet of Things: Vision

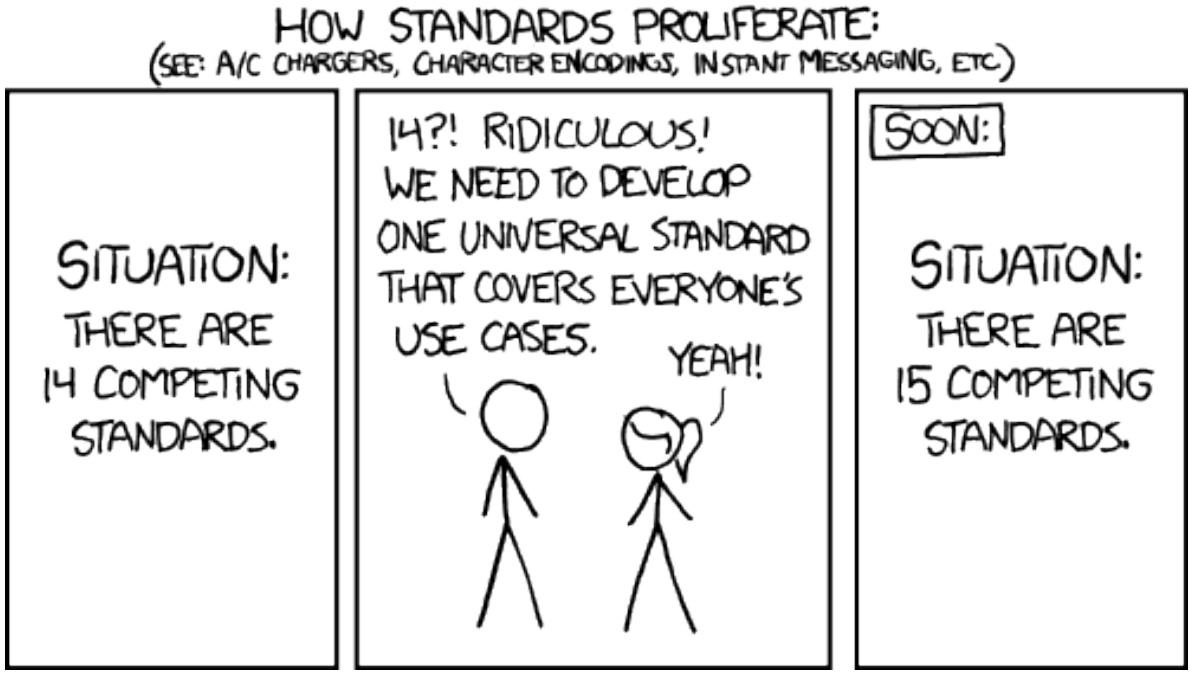
- inter-connected “smart” devices
  - industrial automation
  - wireless sensor networks
  - vehicular communication
  - monitoring and alerting
  - agriculture
  - manufacturing
  - home automation
  - entertainment
  - telemedicine
  - transportation
  - web services
- machine-to-machine communication

Internet of Things: Vision

- inter-connected “smart” devices
  - industrial automation
  - wireless sensor networks
  - vehicular communication
  - monitoring and alerting
  - agriculture
  - manufacturing
  - home automation
  - entertainment
  - telemedicine
  - transportation
  - web services
- machine-to-machine communication
- high-speed low-latency wireless networks

Internet of Things: Vision

- inter-connected “smart” devices
  - industrial automation
  - wireless sensor networks
  - vehicular communication
  - monitoring and alerting
  - agriculture
  - manufacturing
  - home automation
  - entertainment
  - telemedicine
  - transportation
  - web services
- machine-to-machine communication
- high-speed low-latency wireless networks
- cloud computing



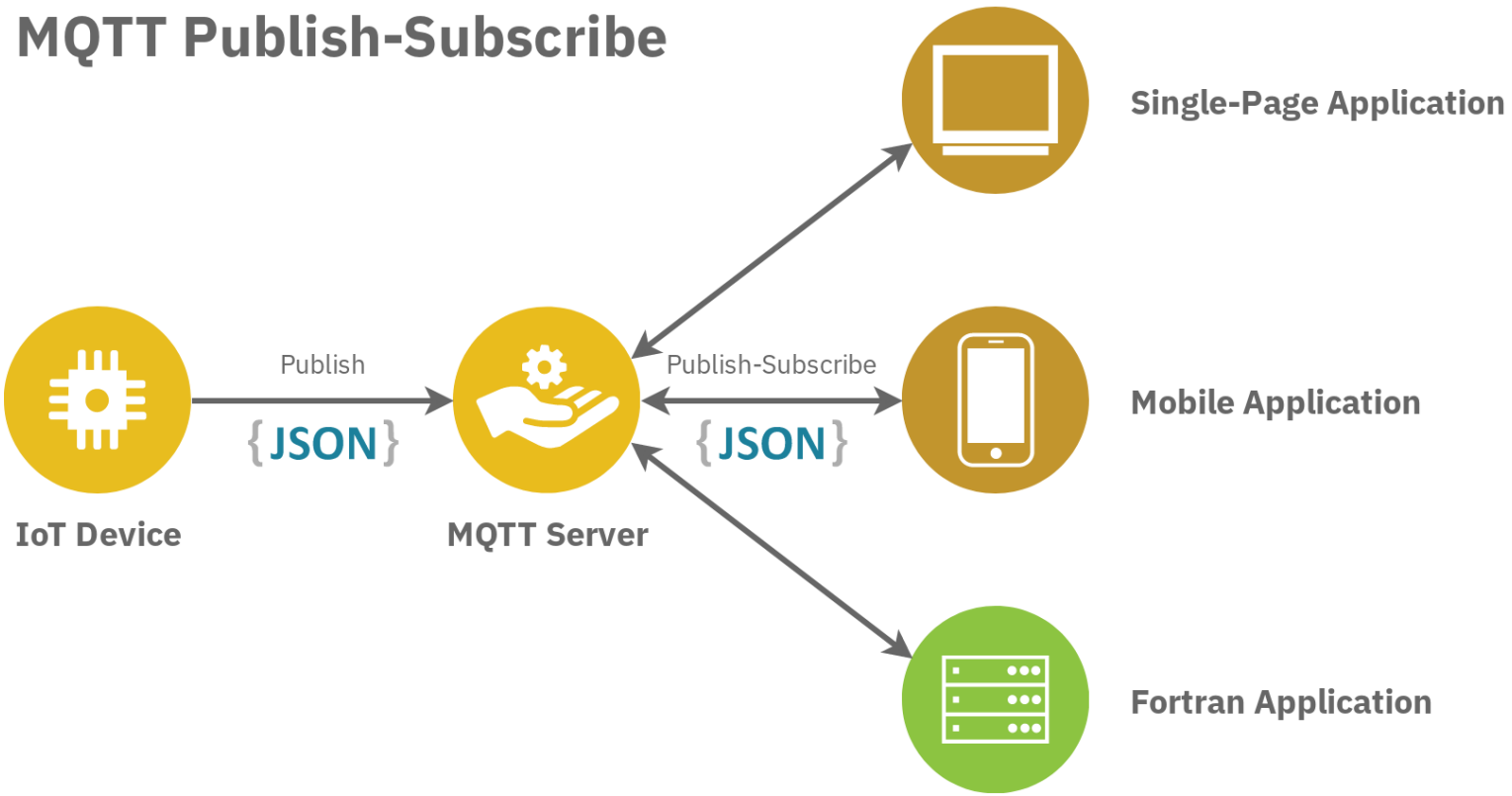
Source: xkcd.com

MQTT

- lightweight client-server protocol for *publish-subscribe* messaging patterns
- message-oriented middleware
- QoS, retained messages
- server bridges
- MQTT-SN for constrained sensor networks
- rich ecosystem[1]

[1] https://github.com/hobbyquaker/awesome-mqtt

MQTT


MQTT Publish-Subscribe
Single-Page Application
Publish
{JSON}
Publish-Subscribe
{JSON}
Mobile Application
IoT Device
MQTT Server
Fortran Application

MQTT Implementations

Client

- Eclipse Paho (C, C++, Java, JavaScript, Python, Go, ...)
- MQTT-C (C)
- luamqtt (Lua)
- PubSubClient (Arduino)
- ...

Server/Broker

- Eclipse Mosquitto (Linux, Unix, macOS, Windows)
- Apache ActiveMQ (Java)
- HBMQTT (Python 3)
- VerneMQ (Linux)
- ...

MQTT and Fortran: Requirements

- Eclipse Paho client library (C)
- *fortran-paho*[2] interface bindings (Fortran 2003)
- Server/Broker (Eclipse Mosquitto)

[2] https://github.com/interkosmos/fortran-paho

MQTT and Fortran: Publishing

```
rc = mqtt_client_create(client, &
                        'tcp://localhost:1883' // c_null_char, &
                        'FortranClient' // c_null_char, &
                        MQTTCLIENT_PERSISTENCE_NONE, &
                        c_null_ptr)

rc = mqtt_client_connect(client, conn_opts)

if (rc == MQTTCLIENT_SUCCESS) then
  pub_msg%payload     = c_loc(payload)
  pub_msg%payload_len = len(payload)

  rc = mqtt_client_publish_message(client, topic, pub_msg, token)
  rc = mqtt_client_wait_for_completion(client, token, TIMEOUT)
  rc = mqtt_client_disconnect(client, TIMEOUT)
end if

call mqtt_client_destroy(client)
```

ZeroMQ

- embeddable messaging kernel library written in C++
- carries atomic messages across multiple transport protocols (in-process, inter-process, TCP, multicast)
  - asynchronous message queues
  - multiple messaging patterns (request-reply, pub-sub, pipeline, exclusive pair)
  - message filtering (subscriptions)
- FZMQ – ZeroMQ bindings for Fortran 2003[3]

[3] https://github.com/richsnyder/fzmq

ZeroMQ

Installation of ZeroMQ 4.3 (FreeBSD):

```
# pkg install net/libzmq4
```

ZeroMQ

Installation of ZeroMQ 4.3 (FreeBSD):

```
# pkg install net/libzmq4
```

Compilation of FZMQ bindings:

```
$ mkdir build && cd build/
$ cmake ..
$ make
```

outputs static library `libzmq.a` and Fortran module `zmq.mod` (link with `-lfzmq` and `-lzmq`)

ZeroMQ and Fortran: Sending Data

```
! sender.f90
program sender
    use, intrinsic :: iso_c_binding
    use :: zmq
    implicit none
    type(c_ptr)     :: ctx, sock
    type(zmq_msg_t) :: msg
    integer         :: n, rc
    real, target    :: pi = acos(-1.0)

    ctx  = zmq_ctx_new()
    sock = zmq_socket(ctx, ZMQ_REQ)
    rc   = zmq_connect(sock, 'tcp://localhost:5555')
    rc   = zmq_msg_init_data(msg, &
                             c_loc(pi), &
                             c_sizeof(pi), &
                             c_null_ptr, &
                             c_null_ptr)
    n  = zmq_msg_send(msg, sock, 0)
    rc = zmq_close(sock)
    rc = zmq_ctx_term(ctx)
end program sender
```

ZeroMQ and Fortran: Receiving Data

```
! receiver.f90
program receiver
    use, intrinsic :: iso_c_binding
    use :: zmq
    implicit none
    character(kind=c_char, len=:), pointer :: buff, rng
    type(c_ptr)                            :: ctx, ptr, sock
    type(zmq_msg_t)                        :: msg
    integer                                :: n, rc
    real                                   :: pi

    ctx  = zmq_ctx_new()
    sock = zmq_socket(ctx, ZMQ_REP)
    rc   = zmq_bind(sock, 'tcp://*:5555')

    rc   = zmq_msg_init(msg)
    n    = zmq_msg_recv(msg, sock, 0)
    ptr  = zmq_msg_data(msg)

    call c_f_pointer(ptr, buff)
    rng => buff(1:c_sizeof(pi))
    pi = transfer(rng, pi)
    rc = zmq_msg_close(msg)
    print *, pi

    rc = zmq_close(sock)
    rc = zmq_ctx_term(ctx)
end program receiver
```

ZeroMQ and Fortran: HTTP Server

- send/receive TCP data to/from non-ZeroMQ peers
- socket type `ZMQ_STREAM`
- either client or server

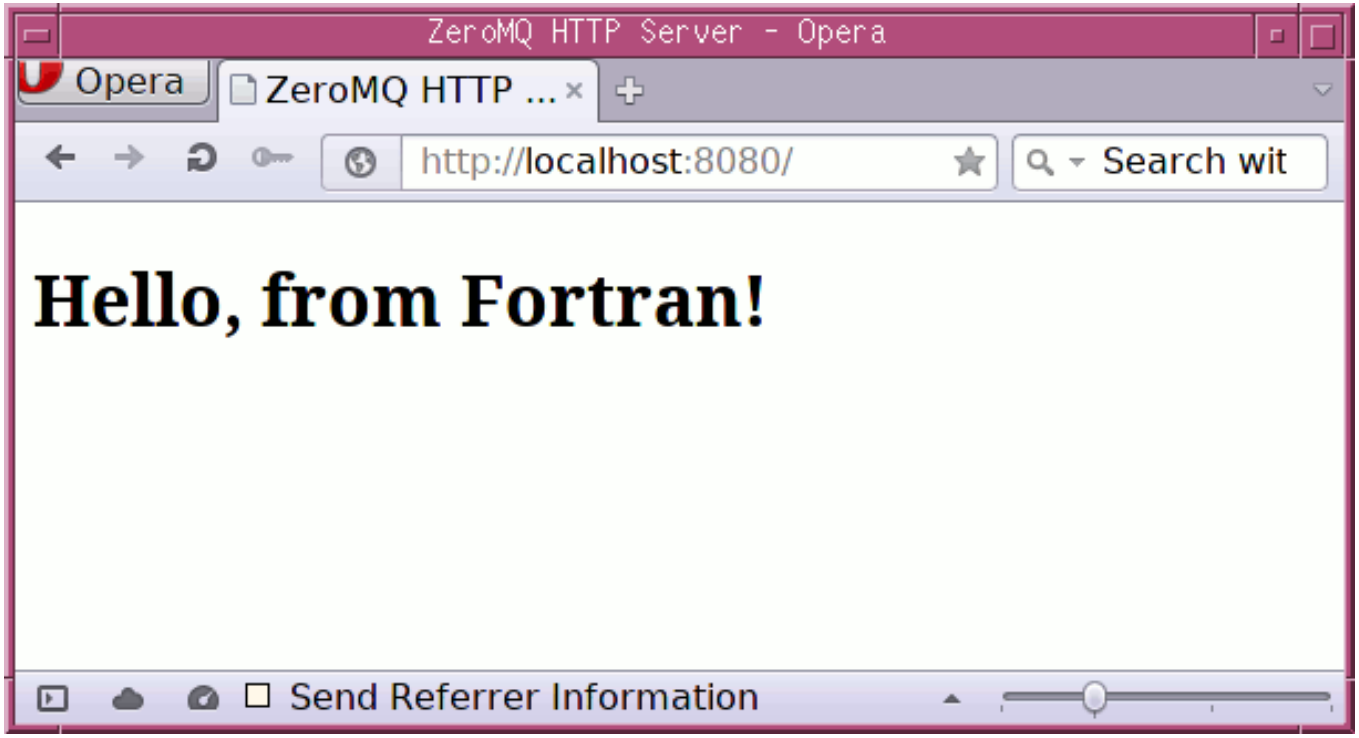


Fig.: ZeroMQ HTTP server in Fortran

NGINX

- open-source web server, reverse proxy, load balancer, ...
- extensible by 3rd party modules[4]
  - embedding Lua, mruby, PostgreSQL, ...
  - *nginx-link-function* module[5] to load shared libraries
  - Fortran 2003 interface bindings *fortran-nginx*[6]
- server-side web applications in Fortran

[4] https://www.nginx.com/resources/wiki/modules/

[5] https://github.com/Taymindis/nginx-link-function

[6] https://github.com/interkosmos/fortran-nginx

NGINX and Fortran: Web Application

```
! webapp.f90
module webapp
    use, intrinsic :: iso_c_binding, only: c_null_char
    use :: ngx_link_func
    implicit none
    logical, save :: is_service_on = .false.
contains
    subroutine ngx_link_func_exit_cycle(cyc) bind(c)
        type(ngx_link_func_cycle_t), intent(in) :: cyc

        is_service_on = .false.
    end subroutine ngx_link_func_exit_cycle

    subroutine ngx_link_func_init_cycle(cyc) bind(c)
        type(ngx_link_func_cycle_t), intent(in) :: cyc

        is_service_on = .true.
    end subroutine ngx_link_func_init_cycle
end module webapp
```

NGINX and Fortran: Web Application

```
! Routine called by nginx on request.
subroutine ngx_hello(ctx) bind(c)
    type(ngx_link_func_ctx_t), intent(in) :: ctx
    character(len=*), parameter           :: response = &
        '<h1>Hello, from Fortran!</h1>'

    call ngx_link_func_write_resp( &
        ctx          = ctx, &
        status_code  = 200_8, &
        status_line  = '200 OK' // c_null_char, &
        content_type = NGX_LINK_FUNC_CONTENT_TYPE_HTML, &
        resp_content = response // c_null_char, &
        resp_len     = int(len(response), kind=8))
end subroutine ngx_hello
```

NGINX and Fortran: Web Application

- Compile and link with *fortran-nginx* static library:

```
$ gfortran -shared -fPIC -o webapp.so \
  webapp.f90 ngx_link_func.a
```

- outputs shared library `webapp.so`

NGINX and Fortran: Server Configuration

nginx.conf:

```
# Load shared library if module is not linked statically:
load_module /usr/local/libexec/nginx/ngx_http_link_func_module.so;

http {
    server {
        listen              80;
        server_name         localhost;
        ngx_link_func_lib /usr/local/etc/nginx/webapp.so;

        location = / {
            ngx_link_func_call "ngx_hello";
        }
    }
}
```

NGINX and Fortran: Web Application

Fig.: Response of `ngx_hello()` in shared library `webapp.so`

NGINX and Fortran: Plotting

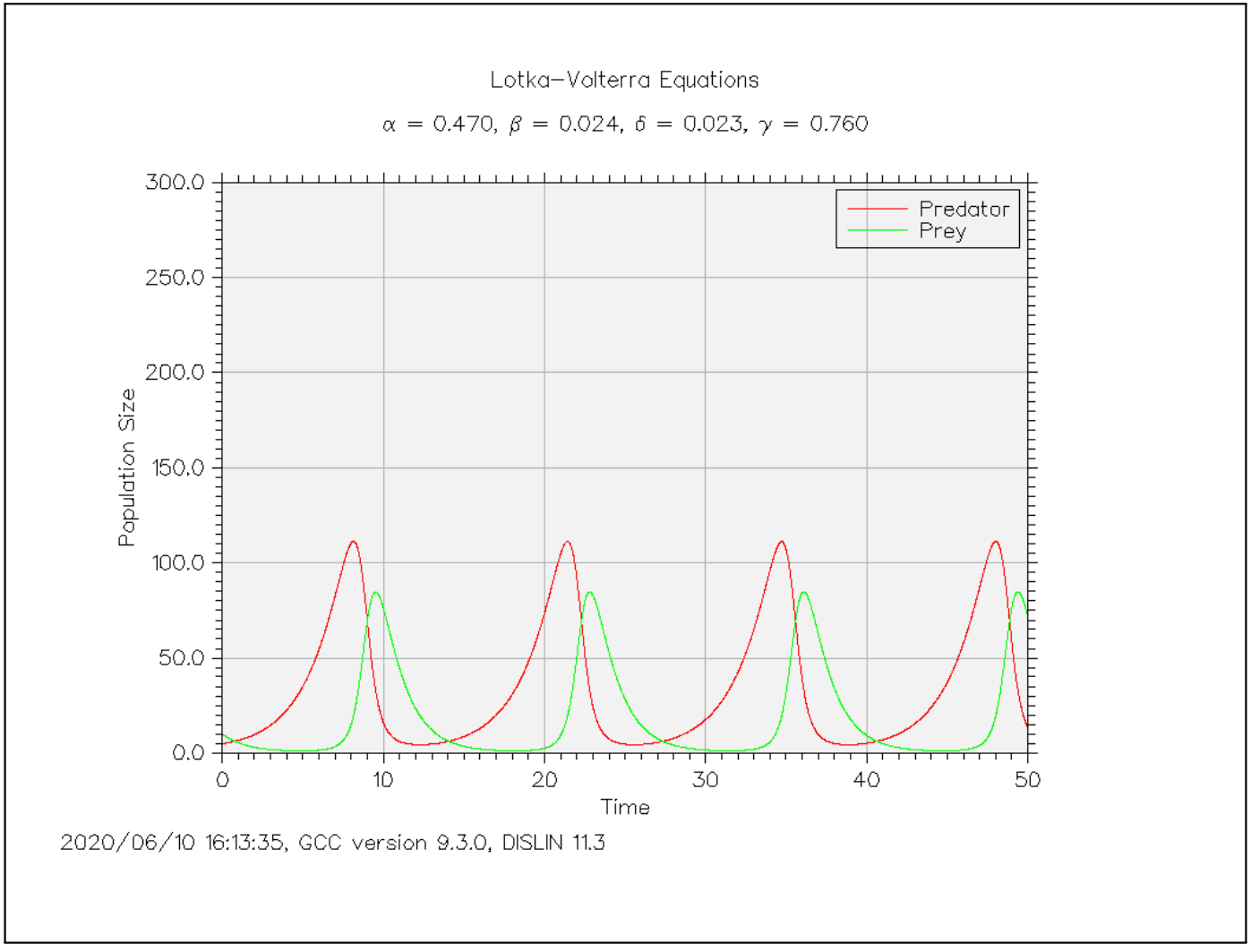


Fig.: http://localhost/plot.png?u=5&v=10

Representational State Transfer (REST)

- HTTP-based architecture for state-less web services
- API access via URIs and HTTP methods (GET, POST, PUT, DELETE, OPTIONS, ...)
- arbitrary payload (JSON, XML, HTML, CSV, ...)
- HTTP Basic Auth, HTTP Digest Auth, JSON Web Token, ...

Representational State Transfer (REST)

```
$ curl http://worldtimeapi.org/api/timezone/Europe/Zurich.txt
abbreviation: CEST
client_ip: XXX.XXX.XXX.XXX
datetime: 2020-06-30T17:40:14.788458+02:00
day_of_week: 2
day_of_year: 182
dst: true
dst_from: 2020-03-29T01:00:00+00:00
dst_offset: 3600
dst_until: 2020-10-25T01:00:00+00:00
raw_offset: 3600
timezone: Europe/Zurich
unixtime: 1593531614
utc_datetime: 2020-06-30T15:40:14.788458+00:00
utc_offset: +02:00
week_number: 27
```

cURL

- client for HTTP, HTTP/2, FTP/TFTP, SMTP, IMAP, Gopher, ...
- cURL command-line tool and library *libcurl*
- `execute_command_line()` (Fortran 2008)
- Fortran 2008 ISO C binding interfaces to *libcurl*[7]

[7] https://github.com/interkosmos/fortran-curl

cURL and Fortran

Retrieving a web resource by URI from Fortran with *fortran-curl*:

```
! http.f90
program http
    use, intrinsic :: iso_c_binding
    use :: curl
    use :: callback
    character(len=*), parameter :: URL = 'http://www.example.com/api/v1/'
    type(c_ptr)                 :: ptr
    integer                     :: rc

    ptr = curl_easy_init()

    rc = curl_easy_setopt(ptr, CURLOPT_URL,            URL // c_null_char)
    rc = curl_easy_setopt(ptr, CURLOPT_TIMEOUT,        int(10, kind=8))
    rc = curl_easy_setopt(ptr, CURLOPT_CONNECTTIMEOUT, int(10, kind=8))
    rc = curl_easy_setopt(ptr, CURLOPT_WRITEFUNCTION,  c_funloc(response_callback))

    rc = curl_easy_perform(ptr)
    call curl_easy_cleanup(ptr)
end program http
```

cURL and Fortran

Callback routine:

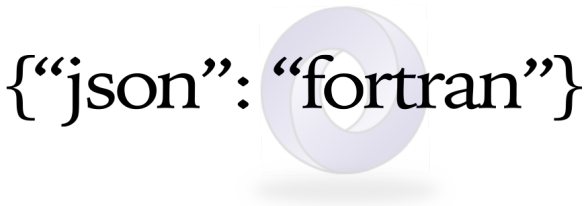
```
! static size_t callback(char *ptr, size_t size, size_t nmemb, void *client_data)
function response_callback(ptr, sz, nmemb, client_data) bind(c)
    type(c_ptr),            intent(in), value :: ptr   ! Chunk of the response.
    integer(kind=c_size_t), intent(in), value :: sz    ! Always 1.
    integer(kind=c_size_t), intent(in), value :: nmemb ! Size of the chunk.
    type(c_ptr),            intent(in), value :: client_data
    integer(kind=c_size_t)                    :: response_callback
    character(len=:), allocatable             :: response

    ! C char pointer to Fortran allocatable character.
    allocate (character(len=nmemb) :: response)
    call c_f_str_ptr(ptr, response)

    write (*, '(a)', advance='no') response
    deallocate (response)

    ! Return number of bytes read.
    response_callback = nmemb
end function response_callback
```

JavaScript Object Notation (JSON)



- lightweight data-interchange format
- derived from the object literals of ECMAScript (RFC 7159)
  - four primitive types (strings, numbers, booleans, null)
  - two structured types (objects, arrays)
- *json-fortran* – object-oriented API for reading and writing JSON data in Fortran 2008[8]

[8] https://github.com/jacobwilliams/json-fortran

JSON and Fortran

example JSON file `data.json`:

```
{
  "sensor": "meteo",
  "temp": 23.1,
  "press": 1011.9,
  "hum": 62.2
}
```

JSON and Fortran

```
program example
    use :: json_module
    implicit none
    real(kind=8)    :: temp, press, hum
    type(json_file) :: json
    logical         :: is_found

    call json%initialize()
    call json%load_file('data.json')

    call json%get('temp',  temp,  is_found)
    call json%get('press', press, is_found)
    call json%get('hum',   hum,   is_found)

    print *, temp, press, hum

    call json%destroy()
end program example
```

Apache CouchDB

- open-source document-oriented NoSQL database
- stores unstructured and semi-structured data in JSON format
- view functions in JavaScript
- RESTful HTTP/JSON API (GET, PUT, POST, DELETE)

Apache CouchDB

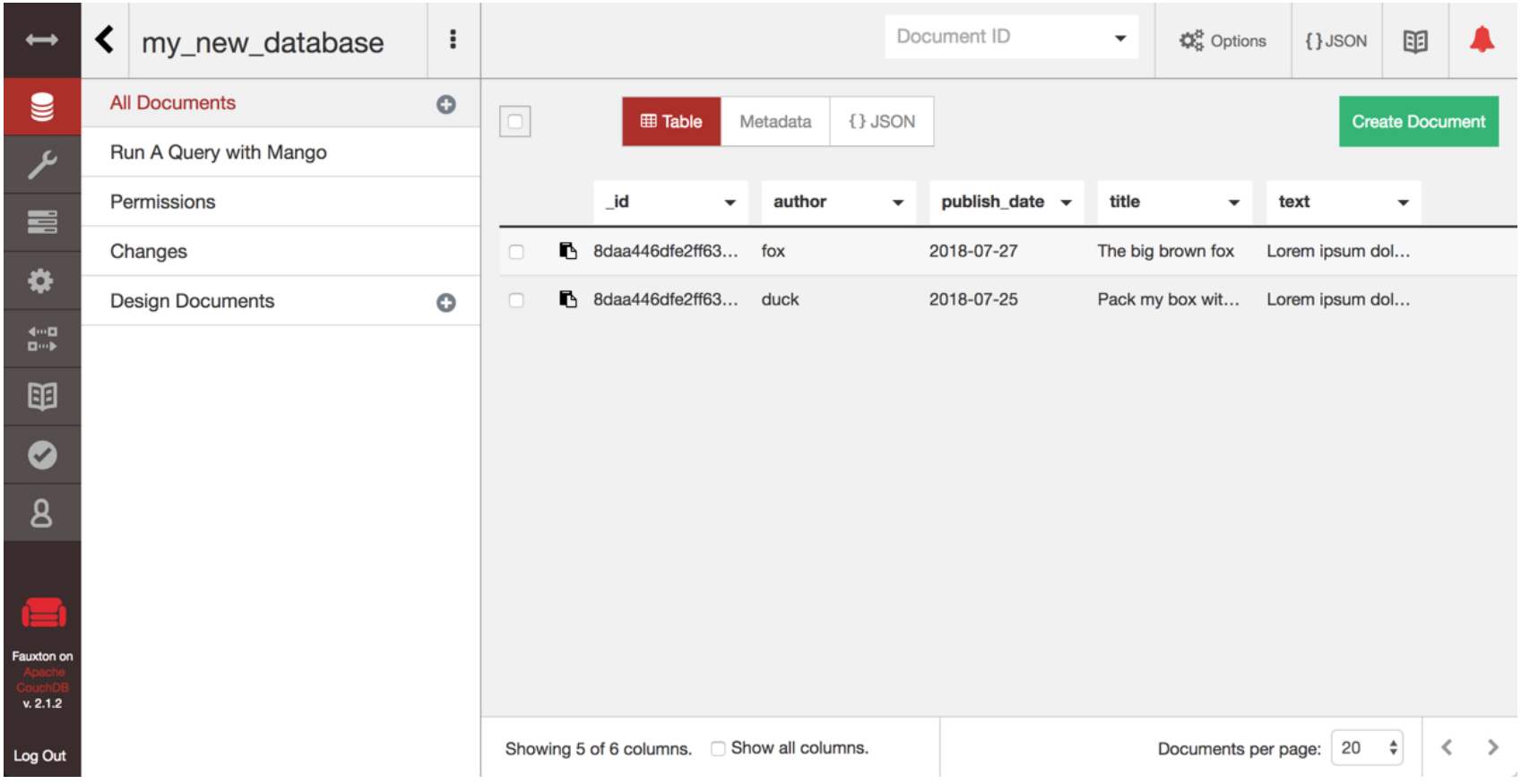


Fig.: Apache CouchDB web interface (Fauxton)

Apache CouchDB

```
$ curl -X GET --user <username>:<password> \
  -G https://couchdb.example.com/timeseries/
{
   "compact_running": false,
   "committed_update_seq": 375048,
   "disk_format_version": 5,
   "disk_size": 33153123,
   "doc_count": 18386,
   "doc_del_count": 0,
   "db_name": "timeseries",
   "instance_start_time": "1290700340925570",
   "purge_seq": 10,
   "update_seq": 375048
}
```

Lua

- embeddable multi-paradigm programming language
- ecosystem of frameworks, libraries, and interfaces[9]
  - string handling, XML, JSON, MessagePack
  - file system access, POSIX, sockets
  - RabbitMQ and Apache Kafka connectivity
  - web frameworks
- target for other languages (MoonScript, Lisp, ...)
- stack-based C API
- Fortran 2003 interfaces to Lua 5.3[10]

[9] https://github.com/LewisJEllis/awesome-lua

[10] https://github.com/interkosmos/fortran-lua53

Calling Lua from Fortran

```
-- script.lua
function hello()
    print('Hello, from Lua!')
end
```

Calling Lua from Fortran

```
-- script.lua
function hello()
    print('Hello, from Lua!')
end
```

```
! example.f90
program main
    use, intrinsic :: iso_c_binding, only: c_ptr
    use :: lua
    implicit none
    type(c_ptr) :: l
    integer     :: rc

    l = lual_newstate()
    call lual_openlibs(l)

    rc = lual_dofile(l, 'script.lua')
    rc = lua_getglobal(l, 'hello')
    rc = lua_pcall(l, 0, 0, 0)

    call lua_close(l)
end program main
```

Calling Lua from Fortran

Compilation of `example.f90` with GNU Fortran:

```
$ gfortran -I/usr/local/include/lua53/ \
  -L/usr/local/lib/lua/5.3/ \
  -o example example.f90 lua.o -llua53
```

Calling Lua from Fortran

Compilation of `example.f90` with GNU Fortran:

```
$ gfortran -I/usr/local/include/lua53/ \
  -L/usr/local/lib/lua/5.3/ \
  -o example example.f90 lua.o -llua53

$ ./example
Hello, from Lua!
```

Calling Fortran from Lua

```
! library.f90
module library
    use, intrinsic :: iso_c_binding
    use :: lua
    implicit none
contains
    function luaopen_library(l) bind(c)
        ! Registers the Fortran routine ‘hello()‘.
        type(c_ptr), intent(in), value :: l
        integer(kind=c_int)            :: luaopen_fortran

        call lua_register(l, 'hello', c_funloc(hello))
        luaopen_fortran = 1
    end function luaopen_fortran

    subroutine hello(l) bind(c)
        ! The Fortran routine callable from Lua.
        type(c_ptr), intent(in), value :: l

        print '(a)', 'Hello, from Fortran!'
    end subroutine hello
end module library
```

Calling Fortran from Lua

Compilation of shared library `library.so`:

```
$ gfortran -fPIC -c lua.f90
$ gfortran -I/usr/local/include/lua53/ \
  -L/usr/local/lib/lua/5.3/ \
  -shared -fPIC -o library.so library.f90 lua.o -llua53
```

Calling Fortran from Lua

Compilation of shared library `library.so`:

```
$ gfortran -fPIC -c lua.f90
$ gfortran -I/usr/local/include/lua53/ \
  -L/usr/local/lib/lua/5.3/ \
  -shared -fPIC -o library.so library.f90 lua.o -llua53
```

Calling Fortran function `hello()` in shared library `library.so` from Lua:

```
-- example.lua
require("library")
hello()
```

Calling Fortran from Lua

Compilation of shared library `library.so`:

```
$ gfortran -fPIC -c lua.f90
$ gfortran -I/usr/local/include/lua53/ \
  -L/usr/local/lib/lua/5.3/ \
  -shared -fPIC -o library.so library.f90 lua.o -llua53
```

Calling Fortran function `hello()` in shared library `library.so` from Lua:

```
-- example.lua
require("library")
hello()
```

Output:

```
$ lua53 ./example.lua
Hello, from Fortran!
```

Python 3

- Python C API
- *F2PY*[11] – generates wrapper function that can be called from Python
- *gfort2py*[12] – calling Fortran code from Python 3
- *forpy*[13], *fortran-python3*[14] – embedding Python in Fortran

[11] http://www.f2py.com/

[12] https://github.com/rjfarmer/gfort2py

[13] https://github.com/ylikx/forpy

[14] https://github.com/interkosmos/fortran-python3


That's all Folks!

Links

GitHub (bindings to MQTT, cURL, NGINX, Lua, Python, ...)
https://github.com/interkosmos

Programming in Modern Fortran on Unix
https://cyber.dabamos.de/programming/modernfortran/