makesense Documentation

Release 0.0.1

Remy Leone

May 14, 2016

Contents

1 User Guide **3**
- 1.1 Installation . . . 3
- 1.2 Quickstart . . . 3
- 1.3 Makesense organization . . . 4
- 1.4 Iotlab . . . 5
- 1.5 Remote server . . . 6
- 1.6 How to make a campaign of simulation? . . . 6

2 Indices and tables **9**

Makesense is a tool that can create, run and analyze WSN experiment. It's designed to be modular.

```
fab new:foobar  # Will create a foobar experiment
fab make:foobar  # Will compile all nodes and simulations tools
fab launch:foobar  # Will run the experiment
fab analyze:foobar  # Will launch all analysis scripts
fab plot:foobar   # Will plot all the graphes
```

CHAPTER 1

User Guide

This part of the documentation, which is mostly prose, begins with some background information about makesense, then focuses on step-by-step instructions for getting the most out of it.

1.1 Installation

Makesense could be set up very easily. It uses git to manage it source code. To get started do the following commands:

```
git clone https://github.com/sieben/makesense.git
```

1.1.1 Dependencies installation

Makesense leverages several Python libraries such as :

- fabric for launching command from the command lines
- networkx for network topology graph analysis
- pandas for managing large datasets
- matplotlib for plotting
- jinja2 for templating

Once you have all the content you should get started by installing all python dependencies. If you have pip installer you can install of them in one hit by using this command:

```
pip install requirements.txt
```

If you prefer to have your python dependencies managed by your package manager on ubuntu for instance you can have :

```
sudo apt-get install fabric python-networkx python-pandas python-matplotlib python-jinja2
```

1.2 Quickstart

This page will walk you through a typical experiment. I suppose that you have installed all the required python libraries.

1.2.1 Configure the Contiki stack

To compile and execute WSN simulation we use the Contiki operating system. We need to set up the fabric file that will be the orchestrator of our experiments to know where is the contiki folder. For that simply put the address of your current contiki in the *fabfile.py* at the root of makesense:

```
# Modify in your fabfile this line
CONTIKI_FOLDER = "/my/path/to/contiki"
```

If you are satisfied with default you don't need to modify anything in the fabfile to get started.

1.2.2 Creating a simulation

Makesense is shipped with a little experiment to quickly get started and get a grasp of what the framework can do. To get started with a dummy experiment simply run the following command in the makesense directory:

```
fab new:dummy  # Will create a dummy experiment
```

This command will create a folder in experiments/dummy containing:

- Contiki source code for server and a client
- A Cooja Simulation Config file (CSC) that will be used to create a simulation
- A Makefile that will help relaunch the

1.2.3 Compiling a simulation

Because makesense know all the path to the Contiki source code and to the simulation file we can simply make all the firmware by typing this command:

```
fab make:dummy
```

This command will compile all the firmware for the dummy experiment and will compile also the simulator if it's not already done.

1.2.4 Launching an experiment

Launching an experiment is pretty straightforward. Simply type:

```
fab launch:dummy
```

The dummy experiment is designed to produce log files that will be analyzed later on. The goal here is to produce as much results as possible.

1.3 Makesense organization

This page is a little description of the organization of makesense.

- docs/ is where all the documentation is stored
- fabfile.py is the fabric command file. This is were the command line tool fab is going to fetch the code it needs
- makesense/ is the folder countaining most of the function doing the work. It's organized as a python package to be able to share variables easily

- templates/ this is where all the templates used during the making of source code are stored.

```
makesense
-- AUTHORS.rst
-- CONTRIBUTING.rst
-- docs
|   -- campaign.rst
|   -- conf.py
|   -- example_fabfile.py
|   -- example_firmware.c
|   -- example_makefile
|   -- index.rst
|   -- installation.rst
|   -- iotlab.rst
|   -- Makefile
|   -- quickstart.rst
|   -- README
|   -- remote.rst
-- fabfile.py
-- LICENSE
-- makesense
|   -- analyze.py
|   -- bootstrap.py
|   -- graph.py
|   -- __init__.py
|   -- make.py
|   -- parser.py
|   -- plot.py
|   -- report.py
|   -- run.py
|   -- sampling.py
|   -- traffic.py
|   -- utils.py
-- README.md
-- requirements.txt
-- templates
    -- dummy_client.c
    -- dummy_main.csc
    -- dummy_makefile
    -- dummy_script.js
    -- dummy_server.c
    -- exp.json
    -- lambda_bootstrap.js
    -- readme.md
    -- report.html
```

1.4 Iotlab

Makesense can perform and deploy code to the iotlab infrastructure.

Let's suppose that you have in your .ssh/config the following

```
Host grenoble
HostName grenoble.iot-lab.info
User leone
```

The you can use commands such as :

fab push_iotlab:dummy # Will launch a experiment fab pull_iotlab:dummy # Will fetch the results of an experiment done on iotlab

1.5 Remote server

There is several tricks and nice features to know when dealing with ssh and remote servers.

Let's suppose that you want to access the grenoble server through ssh. It would be really easy to type just ssh grenoble for instance and get it working. Good news, ssh can do that! Just write in your ~/.ssh/config the following snippet:

```
Host grenoble
HostName grenoble.iot-lab.info
User leone
```

Don't forget to replace leone with your real user name ;-) Then simply do a ssh-copy-id grenoble enter your password and you are all set!

Next you can connect to the server in Grenoble by simply typing ssh grenoble.

An other feature is that it allows to simplify the fabric file. For instance let's suppose that you want to download all the content of a precise folder on your local machine. You can use snippet in your fabric file such as:

```
from fabric.api import env
env.use_ssh_config = True

@task
@hosts("grenoble")
def push(name):
    path = pj(EXPERIMENT_FOLDER, name)
    put(path,
        remote_path="~/html/results/")
```

For more information just check the fabric documentation

1.6 How to make a campaign of simulation?

A campaign of simulation usually involve making a variable change through several run or several nodes.

Makesense leverages Jinja2 to create very easily as many firmware as necessary by using templating. Instead of writing a C code directly we will write C code replacing the variables and function by templates variables that jinja2 will remplace by variables existing in a python code. By doing so we can for instance create very easily a loop iterating through a list of desired values for a variable and let makesense handle all the trouble of creating those files, compile them and deploy them on a testbed.

1.6.1 Step 1: Creating a C template

First we will create a simple C code that print a message through a loop.

```
#include <stdio.h>
#include "contiki.h"

static int my_value = {{ my_value }};

/*---------------------------------------------------------------------------*/
PROCESS(dummy_hello_world_process, "Hello world process");
```

```
AUTOSTART_PROCESSES(&dummy_hello_world_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(dummy_hello_world_process, ev, data)
{
  PROCESS_BEGIN();

  while(1) {
    printf("Hello, world. My value is %d\n", my_value);
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```
static int my_value = {{ my_value }};
```

It's at this place that jinja2 will put the *my_value* variable. For more information check out the Jinja2 documentation.

This makefile will compile all the nodes.

```
SRC=$(wildcard [!symbols]*.c)
PROGS = $(patsubst %.c,%,$(SRC))
all: $(PROGS)

CONTIKI={{ contiki }}
TARGET={{ target }}

include $(CONTIKI)/Makefile.include

# vi:filetype=make:ts=4:sw=4:et
```

1.6.2 Step 2: Let's loop

Suppose that you want to create firmware for 42 different values we would do it in the fabfile:

```
import os
import json
from os.path import join as pj

from fabric.api import task
from jinja2 import Environment, FileSystemLoader

# Default values
ROOT_DIR = os.path.dirname(__file__)
EXPERIMENT_FOLDER = pj(ROOT_DIR, "experiments")
TEMPLATE_FOLDER = pj(ROOT_DIR, "templates")
TEMPLATE_ENV = Environment(loader=FileSystemLoader(TEMPLATE_FOLDER))

@task
def my_special_function(name):
    """ This function will create 42 C files. """
    path = pj(EXPERIMENT_FOLDER, name)
    if not os.path.exists(path):
        os.makedirs(path)

    c_template = TEMPLATE_ENV.get_template("dummy_template.c")
    # We make the id start at 1 and finish at 42
```

```
    for value in range(1, 43):
        with open(pj(path, "dummy_%d.c" % value), "w") as f:
            f.write(c_template.render(my_value=value))

    # If you change the platform target and want to push to iotlab
    # don't forget to update the nodes names
    makefile_template = TEMPLATE_ENV.get_template("dummy_makefile")
    with open(pj(path, "Makefile"), "w") as f:
        f.write(makefile_template.render(contiki=CONTIKI_FOLDER,
                                         target="iotlab-m3"))

    config_template = TEMPLATE_ENV.get_template("dummy_iotlab.json")
    res = [
        {"nodes": ["m3-%d.grenoble.iot-lab.info" % num],
         "firmware_path": pj(path, "dummy_%d.iotlab-m3" % num)
       } for num in range(1, 43)]
    with open(pj(path, "iotlab.json"), "w") as f:
        f.write(json.dumps(res, sort_keys=True,
                  indent=4, separators=(',', ': ')))
```

Then we would call this function like any other fabric function

```
fab my_special_function:dummy
```

You should have files like:

- dummy_1.iotlab-m3
- ...
- dummy_42.iotlab-m3

created in experiments/dummy

You should also have an iotlab.json looking like:

```
[
    {
        "firmware_path": "/home/sieben/Dropbox/workspace/makesense/experiments/prout/dummy_1.iotlab-m
        "nodes": [
            "m3-1.grenoble.iot-lab.info"
        ]
    },
    ...
]
```

1.6.3 Step 3: Push to iotlab

Then we simply have to push to iotlab:

```
fab push_iotlab:dummy
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search