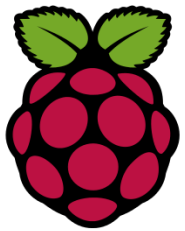
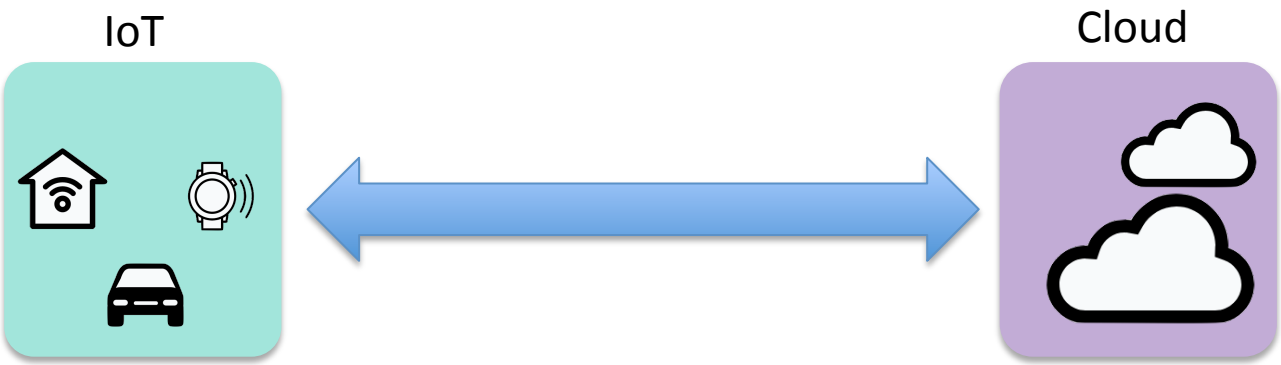
Automating RasPi Installations with Ansible



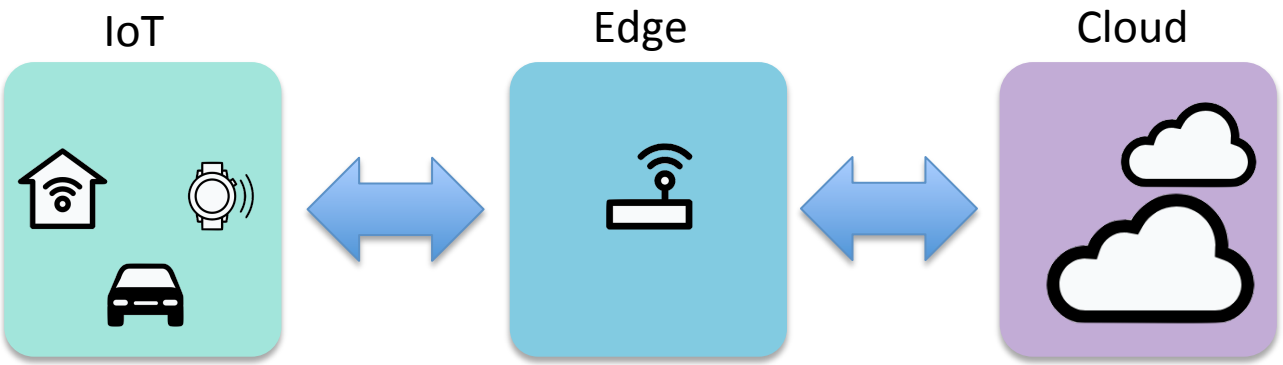
pierre-guillaume.raverdy@inria.fr

Context



IoT devices pushing lots of data to the cloud to be analyzed

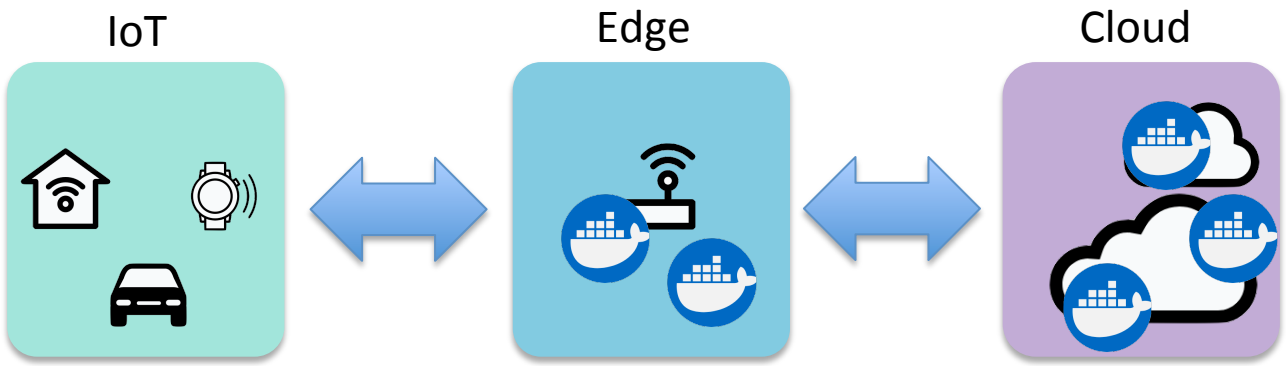
Context



IoT devices pushing lots of data to the cloud to be analyzed

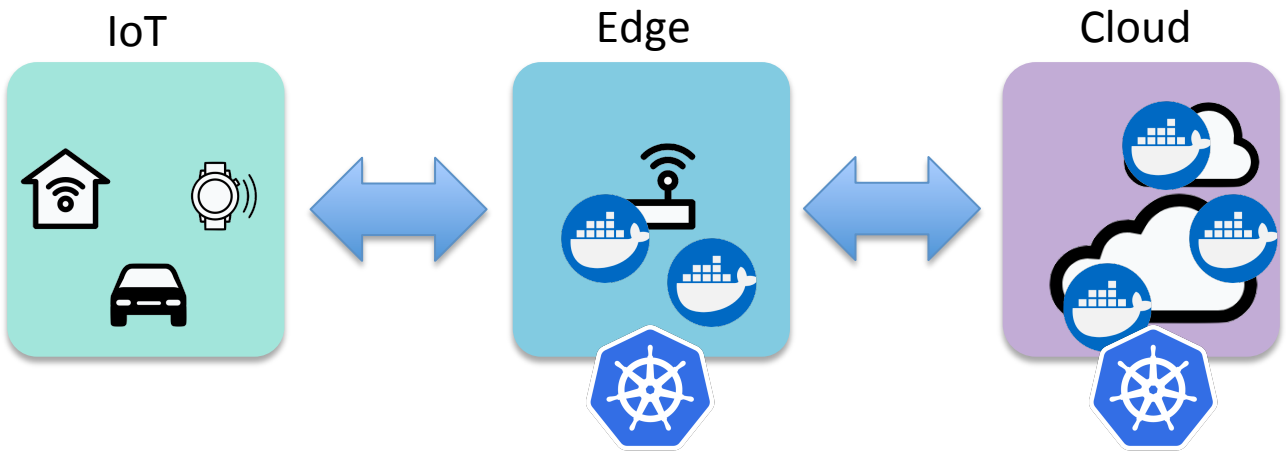
Use gateways at the edge to filter the data (e.g., 4k camera)

Context



Deploy containers at the edge to process data closer to the source

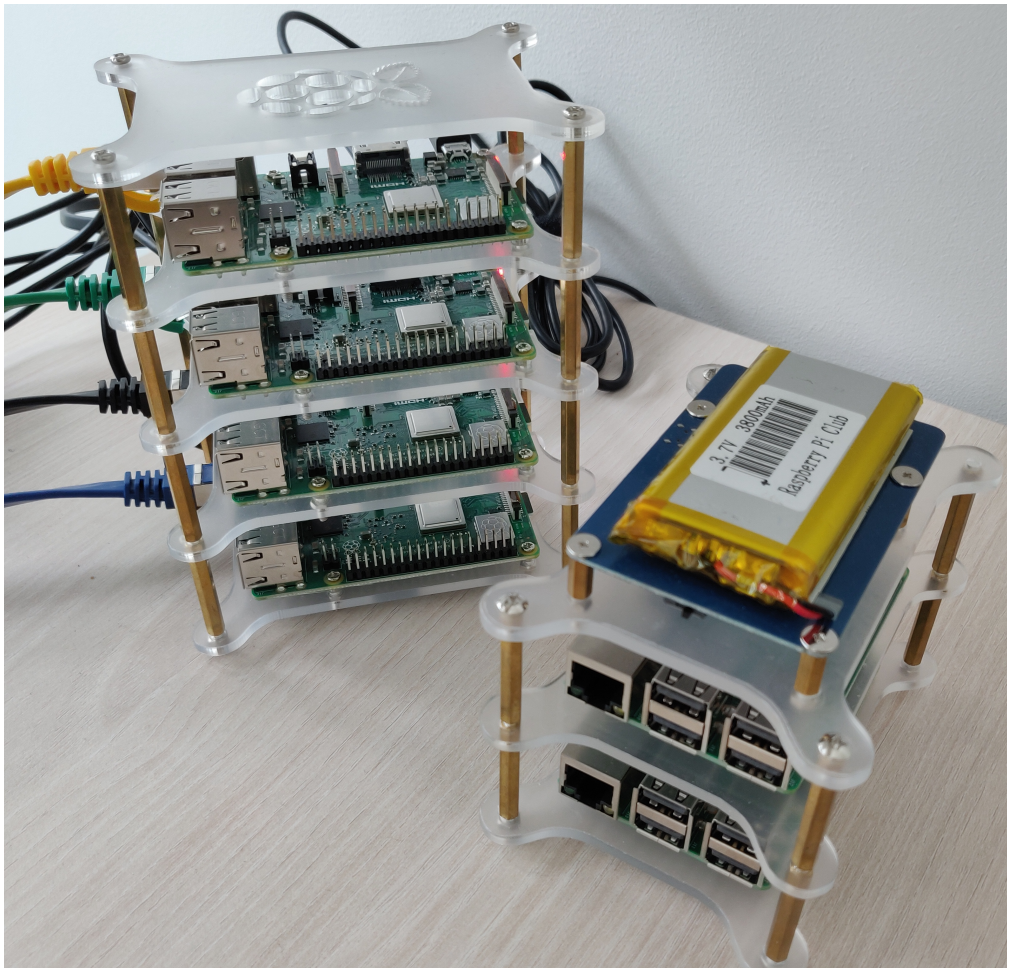
Context



Deploy containers at the edge to process data closer to the source

Orchestrate container deployment to scale, handle connectivity changes

A RaspberryPi testbed



Testbed setup process

1. Flash micro-SD card, boot, initial linux configuration
2. Update/upgrade system, install extra packages
3. Setup network
4. Configure system services
5. Install and setup docker
6. (try to) Install and setup Kubernetes --> step 1

Much **repetition** and **frustration**:

- Full reinstalls before having the right setup,
- Need to do all the Pi's of the setup

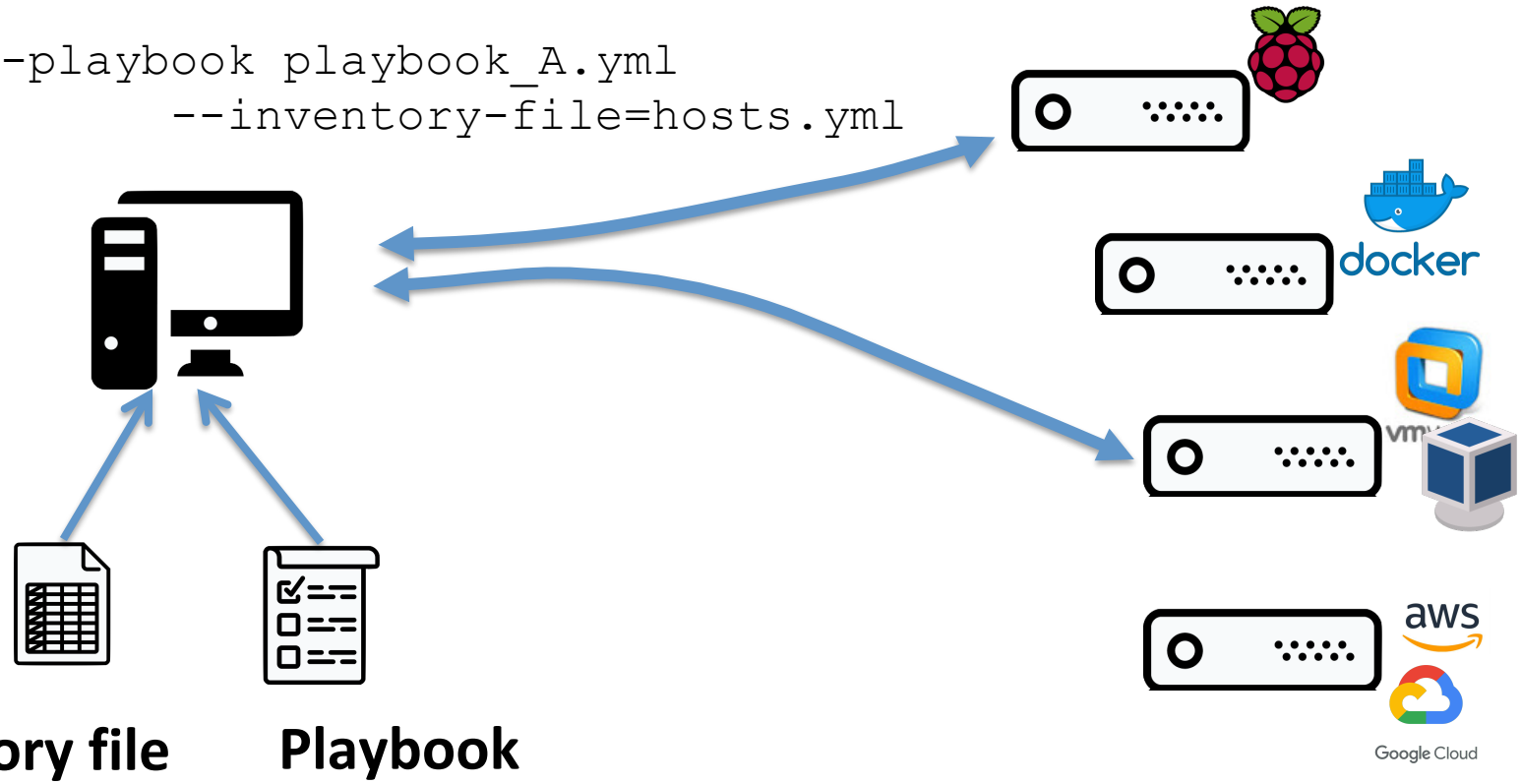
Can't do ssh/interactive sessions all the time

How to automate the setup process?

- Various configuration tools
  - Puppet, Chef, Ansible, Saltstack, …
- Why Ansible?
  - No requirement on the target side (except ssh)
  - Easy to use syntax (YAML)
  - Push-based, idempotent
  - Extensible (thriving community)

How it works ?

```
ansible-playbook playbook_A.yml
                 --inventory-file=hosts.yml
```



Inventory file
defines hosts, groups, variables

Playbook
defines tasks to perform on host/ groups, workflow, conditions, …

Ansible main concepts

Concepts

- Inventory file: definition of hosts, groups, variables
- Playbook: contains plays, tasks
- Modules: plugins that wrap commands, apps and can be used in a task

Workflow features

- Conditional, loops,
- forks, async, pipeline
- Facts, results, tags
- Jinga2 templating, regexp
- Handlers (tasks triggered by notifications, executed at end of play)

Inventory file

Group "all" defines two sub-groups (master, nodes)

Variables can be associated to any level

(variables can be moved in group_vars folder)

```
all: # keys must be unique
  children:

    # master group
    master:
      hosts:
        pimaster1.local:
          static_ip_addr: '192.168.0.70'
          static_ip6_addr: 'fd51:42f8:caae:70::ff'

   # nodes group
    nodes:
      hosts:
        pinode2.local:
          static_ip_addr: '192.168.0.71'
          static_ip6_addr: 'fd51:42f8:caae:71::ff'
      vars:
         node_gateway: '192.168.0.1'

  # all variables
  vars:
    node_interface: 'eth0'

    ls_fliter_var: 'foo.txt'
    somelist: ['ip1','ip2','ip3']

    ansible_connection: ssh
    ansible_user: pi
    ansible_ssh_private_key_file: ./keys/cluster_a_rsa
```

Playbook (2 Plays each with 2 Tasks)

Each play is for a group of hosts (master, nodes)

Tasks call the modules yum and service

```
---
- hosts: master
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- hosts: nodes
  remote_user: root
  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

Had to solve many problems….

Run as root…or as another user

Install, reboot, continue install

Set/Replace lines in configuration files

Set global or host specific values

Extract token/secret key from console output

….found solutions very quickly

Example:
adding/updating a block in a file

Using `blockinfile` module to set or replace text in a file

```
- name: Insert eth def in dhcpcd file
  blockinfile:
    path: /etc/dhcpcd.conf
    marker: "# {mark} ANSIBLE MANAGED BLOCK "
    content: |
          interface {{node_interface}}
          static ip_address={{static_ip_addr}}/24
          static ip6_address={{static_ip6_addr}}/64
          static routers={{node_gateway_inria}}
          static domain_name_servers={{local_dns_inria}} 8.8.8.8
  become: true
```

Marker to delimit bloc to set/replace

Run task as root

Task example: loops & conditionals

Execute the `ls` shell command, and **store the result in the `waiting_for_ls` variable**

Loop until result contains variable. Try at most 10 times, every 10 seconds.

```
- name: Wait for a file to be created on the master node.
  hosts: master
  remote_user: pi
  tasks:
    - name: Wait for ls to match (need to create foo.txt file)
      shell: 'ls -la'
      # register the result of the command in a variable
      register: waiting_for_ls
      until: (ls_fliter_var in waiting_for_ls.stdout)
      retries: 10
      delay: 10
```

Waits for foo.txt to be created on the filesystem

Task example: reboot

Launch task in async mode (fire and forget) and continue play. Do task if previous task done also.

Use `wait_for_connection` module to wait for host, and continue play afterwards

Can also use *`ansible_job_id`* to check on status of async task

```
- name: create test file
    shell: touch rebootlog.txt
    register: task_result

- name: display time before reboot
    # shell command to add a line for each item in the loop data
    shell: echo 'before reboot' >> rebootlog.txt; date >> rebootlog.txt

- name: Reboot the server and wait for it to come back up.
    shell: sleep 5 && reboot
    async: 1
    poll: 0
    when: task_result is changed
    become: true

- name: Wait for the reboot to complete if there was a change.
    wait_for_connection:
      connect_timeout: 20
      sleep: 5
      delay: 5
      timeout: 300
    when: task_result is changed

- name: display time after reboot
    # shell command to add a line for each item in the loop data
    shell: echo 'after reboot' >> rebootlog.txt; date >> rebootlog.txt
```

Conclusion

First impressions:

- Very easy to learn & use
- Good documentation, community
- Useful for automation and documentation

Like Dockerfile