paradrop Documentation

Release 0.9.2

Paradrop Labs

Oct 09, 2017

Contents

This section introduces the background and motivation of ParaDrop.

CHAPTER 1

Cloud computing vs. edge computing?

Cloud computing platforms, such as Amazon EC2, Microsoft Azure and Google Cloud Platform have become a popular approach to provide ubiquitous access to services across different user devices. Cloud computing is beneficial for infrastructure providers, service providers, and end users. Infrastructure providers, i.e., cloud platform providers, take advantage of the economies of scale by managing and operating resources in a centralized manner. Cloud computing also provides reliable, scalable, and elastic resources to service providers. In addition, end users can access high-performance computing and large storage resources anywhere with Internet access at anytime thanks for the cloud computing.

Cloud computing requires developers to host services and data on off-site data centers. That means the computing and storage resources are spatially distant from end-users. However, a growing number of high-quality services desire computational tasks to be located nearby. These services desire for lower latency, greater responsiveness, a better end-user experience, and more efficient use of network bandwidth.

ParaDrop is a research effort to build an edge computing platform. The service deployment process of ParaDrop is like "paradrop" some resources from military bases into the battlefields on-demand. In terms of ParaDrop, users can "paradrop" edge computing services (chutes, short of parachutes) from the cloud on-demand. And the whole process is transparent to end users. Based on previous research work on exploring advantages of edge computing, we focus on building a user-friendly edge computing platform for both normal users and developers.

CHAPTER 2

Where is the vantage point for edge computing?

There are various options to place edge computing nodes. ParaDrop chooses to place the edge computing substrate in the WiFi access points (APs). The WiFi AP is a unique platform for edge computing because of multiple reasons: it is ubiquitous in homes and enterprises, is always “on” and available, and it sits directly in the data path between Internet resources and the end users.

CHAPTER 3

How to design an efficient, effective, and flexible edge computing platform?

ParaDrop provides a similar runtime environment as the cloud computing platform to developers, so that developers can easily port some parts of their services from the cloud to the ParaDrop to take advantages of edge computing. Through a lightweight virtualization technique based on docker, ParaDrop provides a flexible environment for developers to build edge computing services with the programming languages, libraries, and frameworks they prefer. ParaDrop offers a well-defined API, that developers can leverage to implement and deploy edge computing services transparent to end users.

CHAPTER 4

System Architecture

This section details some of the non-obvious architectural features of ParaDrop.

ParaDrop has three major components: the ParaDrop router, the ParaDrop controller, and the ParaDrop API.

ParaDrop router

The ParaDrop router is the key part of the ParaDrop platform. In addition to a normal WiFi access point, it provides the substrate to deploy edge computing services on. We have built the reference ParaDrop routers based on the PC Engines APU and APU2 single board computer . The image below shows a router built with a PC Engines APU board.

The ParaDrop software implementation does support various hardware platform. We can also try the functions of ParaDrop with virtual machines. Please visit the *ParaDrop Devices* page for more information about hardware setup for ParaDrop routers.

ParaDrop controller

The ParaDrop controller is deployed at paradrop.org. Users can sign up to create an account through the web page. For end users, it provides a dashboard to configure and monitor the ParaDrop routers they have permissions. They can also manage the edge computing services (we call them chutes, in short of parachutes) running on the ParaDrop routers. For developers, it provides the interface that developers can register their applications and manage the chutes.

ParaDrop API

ParaDrop exports the platform's capability through an API. Based on the functionality and position, the API can be divided into two parts: the cloud part and the edge part. The cloud part provides the management interfaces for applications to orchestrate the chutes from the cloud. Examples include resource permission management, chute deployment and management, router configuration management, etc.. The edge part exports the local context information of the routers to the chutes to do some useful things locally. Examples include local wireless channel information, local wireless peripheral device access, etc..

CHAPTER 5

ParaDrop Devices

This section describes various hardware platforms that we support for running ParaDrop.

If this is your first time working with ParaDrop and you do not have access to any of the supported hardware platforms, we recommend starting with our pre-built virtual machine image.

With the hardware platform up and running, you are ready to activate the router in ParaDrop. The page *Get Started* gives detailed information about that.

Contents:

Virtual Machine

This will quickly take you through the process of bringing up a Hello World chute in a virtual machine on your computer.

NOTE: These instructions assume you are running Ubuntu. The steps to launch a virtual machine may be different for other environments.

Environment setup

These steps wil download our router image and launch it a virtual machine.

1. Install required packages:

```
sudo apt-get install qemu-kvm
```

2. Download the latest build of the Paradrop disk image. https://paradrop.org/release/latest/paradrop-amd64.img.xz

3. Extract the image:

```
xz -d paradrop-amd64.img.xz
```

4. Launch the VM:

```
sudo kvm -m 1024 \
-netdev user,id=net0,hostfwd=tcp::8000-:8000,hostfwd=tcp::8022-:22,
↪hostfwd=tcp::8080-:80 \
-device virtio-net-pci,netdev=net0 -drive file=paradrop-amd64.img,format=raw
```

First Boot Setup

After starting the virtual machine for the first time, follow the instructions on the screen. When it prompts for an email address, enter *info@paradrop.io*. This sets up a user account on the router called *paradrop* and prepares the router to receive software upgrades. Allow the router 1-2 minutes to complete its setup before proceeding.

Please note: there is no username/password to log into the system console. Please follow the steps in the next sections to access your router through paradrop.org or through SSH.

Connecting to your Router with SSH

The router is running an SSH server, which is forwarded from localhost port 8022 with the kvm command above. The router does not accept password login by default, so you will need to have an RSA key pair available, and you can use the router configuration page to upload the public key and authorize it.

1. Open tools page on the router (http://localhost:8080/#!/tools).
2. Find the SSH Keys section and use the text area to submit your public key. Typically, your public key file will be found at *~/.ssh/id_rsa.pub*. You can use *ssh-keygen* to generate one if you do not already have one. Copy the text from the file, and make sure the format resembles the example before submitting.
3. After the key has been accepted by the router, you can login with the command *ssh -p 8022 paradrop@localhost*. The username may be something other than paradrop if you used your own Ubuntu One account instead of info@paradrop.io during the First Boot Setup.

Managing Virtual Machines Using virt-manager

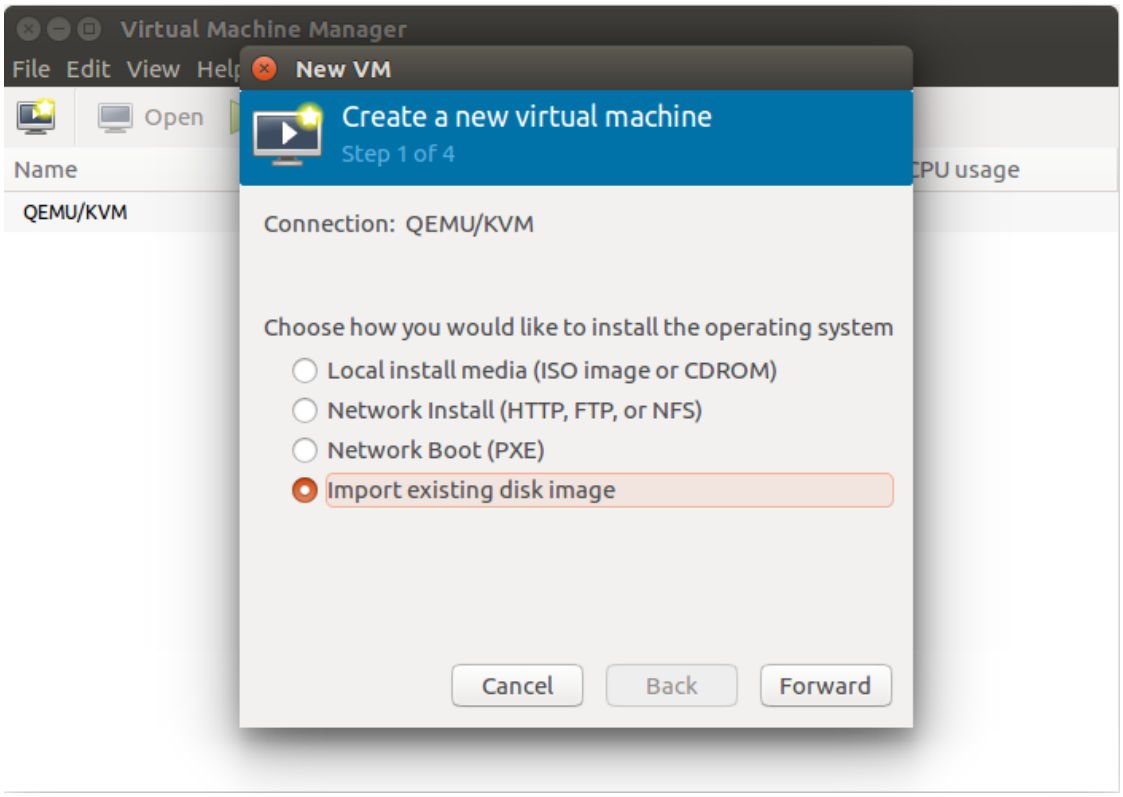
Even though many developers prefer command line tools to manage virtual machines, some developers likes to use GUI tools. In addition, GUI tools are convenient to support some advanced features, e.g., assigning some peripheral devices (USB WiFi dongle) from host to virtual machines. We recommend using "virt-manager" to run ParaDrop virtual machines. If you have not installed it on Ubuntu, you can use below command to install it:

```
sudo apt-get install virt-manager
```

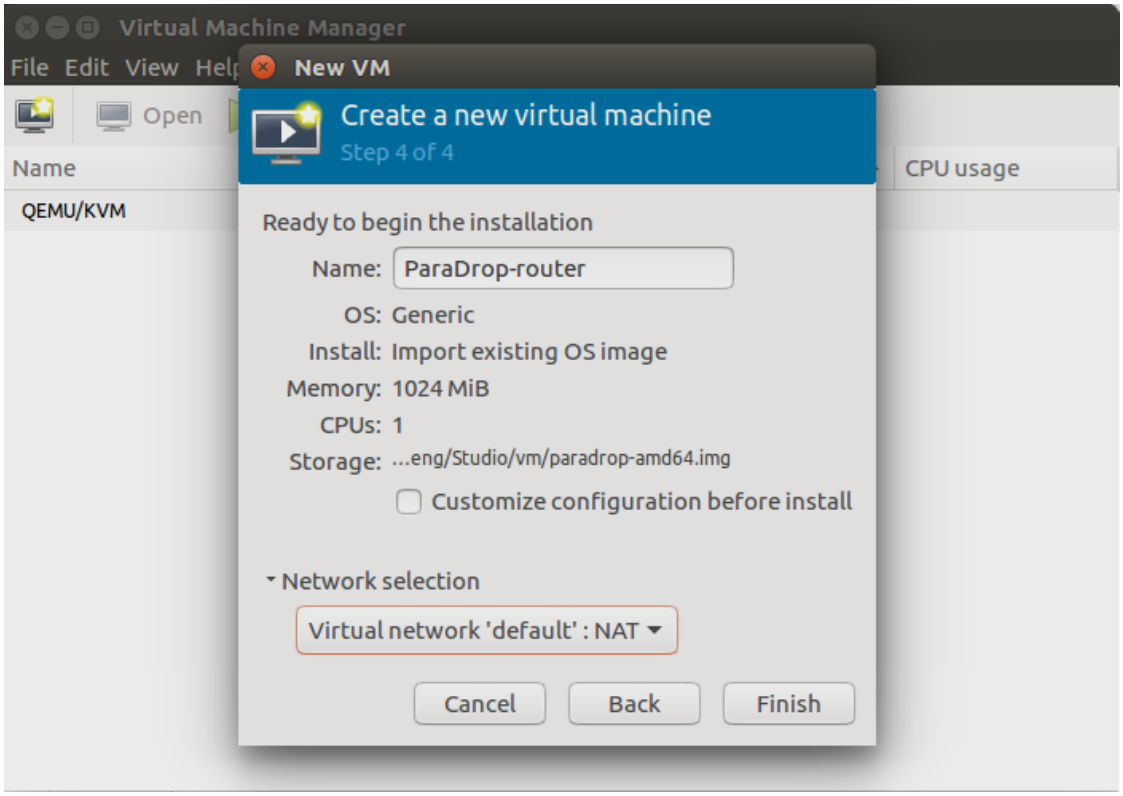
Then we can start virt-manager with below command:

```
sudo virt-manager
```

We can create a VM with the ParaDrop disk image.



Below is the configuration of the VM.



After that, we can boot the VM and configure the first boot as we do when run the VM with command line tools. However, the VM will have an IP address 192.168.122.x, so we can access http://<IP address of the VM> to access

the portal to upload ssh keys, and then login to it directly with the IP address.

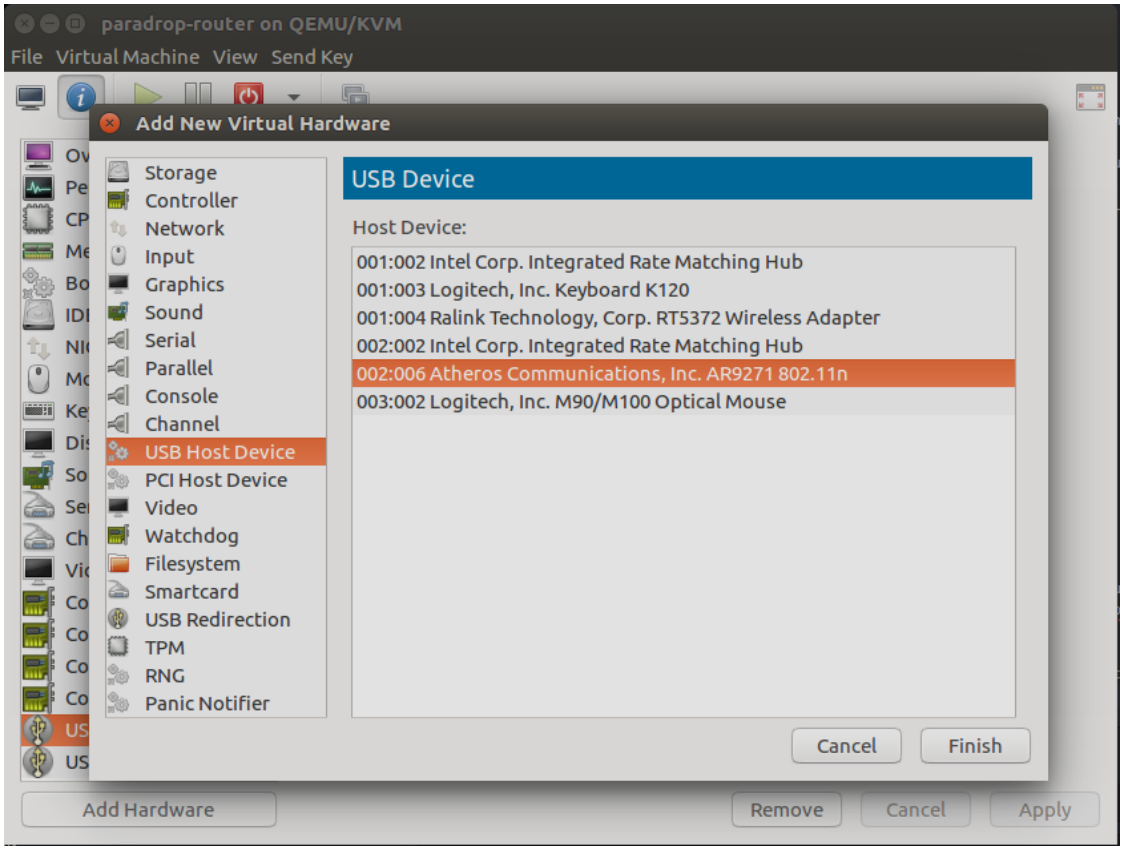
We can assign the USB WiFi dongle from the Host to the ParaDrop VM so that the ParaDrop running on the VM can support features related to WiFi. Before we do that, we need to disable the WiFi device for Host. We can do that with "rflist" command. Run below command to get the number of the WiFi device:

```
rflist list
```

Suppose the number of the WiFi device we want to assign to the ParaDrop VM is 2, then run below command to disable it for host OS:

```
rflist block 2
```

Then we can add the USB WiFi dongle to the VM.



We can run below command in ParaDrop VM to verify that the WiFi device has been detected:

```
iw dev
```

Sometimes, we have to repeat above steps to make sure the WiFi device can be used by the ParaDrop VM.

PC Engines APU2

Hardware requirements

- 1x system board (apu2c4)

- 1x case (case1d2u)
- 1-2x miniPCIe Wi-Fi modules (wle200nx for 802.11n or wle600vx for 802.11ac)
- 2-4x pigtails (pigsma)
- 2-4x antennas (antsmadb)
- 1x power supply (ac12vus2)
- 1x storage module (sd4b) or alternative (see below)

Storage Module

The APU can boot from an SD card or an m-SATA SSD. These instructions are written assuming you will use an SD card because they are easier to flash from another machine. However, we do frequently build Paradrop routers with SSDs to take advantage of the higher storage capacity and read/write speeds. The 4GB pSLC module listed above is known to be very reliable, but you may also prefer a larger SD card.

Preparing the SD card

1. Download the latest build of the Paradrop disk image. https://paradrop.org/release/0.9/paradrop-amd64.img.xz
2. Insert the SD card into the machine you used to download the image and find the device node for the card. This is often "/dev/sdb", but please make sure, as the next command will overwrite the contents of whatever device you pass.
3. Copy the Paradrop image to the SD card.:

```
xzcat paradrop-amd64.img.xz | sudo dd of=<DEVICE> bs=32M status=progress; sync
```

4. Remove the SD card and proceed to assemble the router.

First Boot

The first time you boot the Paradrop router, you will need to connect a serial cable to complete the installation process through a console. The default configuration is 9600 8N1. After the router boots, press Enter when prompted and follow the instructions on the console to configure Ubuntu Core. If you have an Ubuntu One account, you can enter the email address here. Otherwise, you may also enter the address "info@paradrop.io". You will be able to manage your router and install chutes through paradrop.org either way.

Take note of the IP address displayed in the console. You will need this address for the next step, activating the router. For example, the message below indicates that the router has IP address 10.42.0.162.

```
Congratulations! This device is now registered to info@paradrop.io.

The next step is to log into the device via ssh:

ssh paradrop@10.42.0.162
```

Intel NUC

These instructions will help you install the ParaDrop daemon on the Intel NUC platform. At the end of this process, you will have a system ready for installing chutes.

We have specifically tested this process on the Skull Canyon (NUC6i7KYK) platform, which we recommend for high performance edge-computing needs.

Hardware and software requirements

- **Intel NUC Skull Canyon NUC6i7KYK**
  - The Intel NUC devices generally do not come with memory or storage pre-installed.
  - Memory: we recommend at least one 8 GB DDR4 SODIMM.
  - Storage: we have generally found one 16 GB SD card to be sufficient for our storage needs, but we recommend using one MX300 M.2 SSD card for the higher read and write speeds.
  - We recommend updating the BIOS on the NUC. Follow the instructions on the Intel support site.
- 2 USB 2.0 or 3.0 flash drives (each 4 GB minimum)
- A monitor with an HDMI interface
- A network connection with Internet access
- An Ubuntu Desktop 16.04.1 LTS image.
- A ParaDrop disk image.

Preparing for installation

1. Download the Ubuntu Desktop image and prepare a bootable USB flash drive.
2. Download the ParaDrop disk image and copy the file to the second flash drive.

Boot from the Live USB flash drive

1. Insert the Live USB Ubuntu Desktop flash drive in the NUC.
2. Start the NUC and push F10 to enter the boot menu.
3. Select the USB flash drive as a boot option.
4. Select "Try Ubuntu without installing".

Flash ParaDrop

1. Once the system is ready, insert the second USB flash drive which contains the ParaDrop disk image.
2. Open a terminal and run the following command, where <disk label> is the name of the second USB flash drive. You may wish to double-check that /dev/sda is the desired destination **before running dd**.

```
zcat /media/ubuntu/<disk label>/paradrop_router.img.gz | sudo dd of=/dev/sda␣
↪bs=32M status=progress; sync
```

3. Reboot the system and remove all USB flash drives when prompted to do so.

First boot

1. At the Grub menu, press 'e' to edit the boot options.
2. Find the line that begins with "linux" and append the option "nomodeset". It should look like "linux (loop)/kernel.img $cmdline nomodeset". Adding this option will temporarily fix a graphics issue that is known to occur with the Intel NUC.
3. Press F10 to continue booting.
4. Press Enter when prompted, and follow the instructions on the screen to configure Ubuntu Core. If you have an Ubuntu One account. By connecting your Ubuntu One account, you will be able to login via SSH with the key(s) attached to your account. Otherwise, if you do not have an Ubuntu One account or do not wish to use it, you may enter "info@paradrop.io" as your email address. You will still be able to manage your router and install chutes through paradrop.org either way.
5. Take note of the IP address displayed on the screen. You will need this address for the next step, activating the router. For example, the message below indicates that the router has IP address 10.42.0.162.

```
Congratulations! This device is now registered to info@paradrop.io.

The next step is to log into the device via ssh:

ssh paradrop@10.42.0.162
...
```

CHAPTER 6

Get Started

This section goes through the steps to create a ParaDrop account, activate a ParaDrop router, and install a hello-world chute on the router.

Create a ParaDrop account

With a ParaDrop account, users can manage the resources of ParaDrop through a web frontend.

1. Signup at https://www.paradrop.org/signup. You will receive a confirmation email from paradrop.org after you finish the signup.
2. Confirm your registration in the email.

Activate a ParaDrop router

Activation associates the router with your account on paradrop.org so that you can manage the router and chutes through a web frontend.

1. Login to paradrop.org if you do not.
2. From the Routers tab, click Create Router. Give your router a unique name and an optional description to help you remember it and click Submit.
3. On the router page, find the "Router ID" and "Password" fields. You will need to copy this information to the router so that it can connect.
4. Open the router portal at http://<router_ip_address> (or http://home.paradrop.org if you are connected to the LAN port of the router or WiFi network). You may be prompted for a username and password. The default login is "paradrop" with an empty password.
5. Click the "Activate the router with a ParaDrop Router ID" button and enter the information from the paradrop.org router page. If the activation was successful, you should see checkmarks appear on the "WAMP Router" and "ParaDrop Server" lines. You may need to refresh the page to see the update.

6. After you activate your router, you will see the router status is online at https://paradrop.org/routers.

Install a hello-world chute

1. Make sure you have an activated, online router.
2. Go to the Chute Store tab on paradrop.org. There you will find some public chutes such as the hello-world chute. You can also create your own chutes here.
3. Click on the hello-world chute, click Install, click your router's name to select it, and finally, click Install.
4. That will take you to the router page again where you can click the update item to monitor its progress. When the installation is complete, an entry will appear under the Chutes list.
5. The hello-world chute starts a webserver, which is accessible at http://<router-ip-address>/chutes/hello-world. Once the installation is complete, test it in a web browser.

CHAPTER 7

How to Contribute

This section focuses on the *ParaDrop Daemon*. This is the set of daemons and tools required to allow the Paradrop platform to function on virtual machines and real hardware. ParaDrop is an open source project. The source code of the ParaDrop daemon is available at github. Issue reports and pull requests are welcome.

Contents:

ParaDrop daemon development

ParaDrop repository includes a set of tools to make development as easy as possible.

Currently this system takes the form of a bash script that automates installation and execution. This page outlines the steps required to manually install the dependencies, build the package and install the package into hardware/VMs.

We recommend using Ubuntu 16.04 LTS as the development environment for this version of ParaDrop. Because we use snapcraft to package and distribute the ParaDrop daemon.

You will only need to follow these instructions if you will be making changes to the ParaDrop daemon. Otherwise, you can use our pre-built ParaDrop snap or disk image from ParaDrop release.

Building ParaDrop daemon

pdbuild.sh is the script we work with during the development. It provides following commands:

- `./pdbuild.sh setup` installs development dependencies.

- `./pdbuild.sh run` executes the ParaDrop daemon locally in the development machine. It is useful for debugging.
- `./pdbuild.sh build` builds the snap package. Check snapcraft documentation for detailed information about snap packages and snapcraft.
- `./pdbuild.sh image` builds the ubuntu core image that we can flash into SD card or SSD module of a ParaDrop router. It pre-install the dependent snaps for us automatically (docker, airshark).

Installing ParaDrop into hardware/VMs

After the ParaDrop daemon snap is ready (paradrop-daemon_<version>_amd64.snap), we can install it into a ParaDrop router. Check *ParaDrop Devices* for information about preparing a ParaDrop router.

Copy the paradrop snap to the router with ParaDrop image installed:

```
scp paradrop-daemon-<version>_amd64.snap paradrop@<router ip>:~/
```

Then we can login a ParaDrop router:

```
ssh paradrop@<router ip>
```

Install the dependent snaps in a ParaDrop router:

```
snap install docker
snap install airshark
```

Install the newly created ParaDrop daemon snap package:

```
snap install paradrop-daemon-<version>_amd64.snap --devmode
```

Checking logs of ParaDrop daemon

After install the ParaDrop daemon, we can use 'pdlog' to check the log of ParaDrop daemon on the ParaDrop router:

```
paradrop-daemon.pdlog -f
```

Building ParaDrop tools

We have published the ParaDrop tools snap in Ubuntu snap store. On the development machine, we can install it with below command:

```
snap install paradrop-tools
```

Get the manual of ParaDrop tools:

```
paradrop-tools.pdtools --help
```

More detailed information about ParaDrop tools can be find in *Develop Applications for ParaDrop*. The git repository of ParaDrop includes the source code of ParaDrop tools. Developers can build the latest version of ParaDrop tools by running below command in the folder 'tools':

```
snapcraft
```

ParaDrop daemon design

TODO

Documentation and tests

Documentation is handled by sphinx and readthedocs.

Testing is a joint effort between nosetests, travis-ci, and coveralls.

Documentation

Sphinx reads files in reStructuredText and builds a set of HTML pages. Every time a new commit is pushed to github, readthedocs automatically updates documentation.

Additionally, sphinx knows all about python! The directives `automodule`, `autoclass`, `autofunction` and more instruct sphinx to inspect the code located in `paradrop/daemon/paradrop/` and build documentation from the docstrings within.

For example, the directive `.. automodule:: paradrop.backend` will build all the documentation for the given package. See Docstring Conventions for details.

All docstring documentation is rebuilt on every commit (unless there's a bug in the code.) Sphinx does not, however, know about structural changes in code! To alert sphinx of these changes, use the `autodoc` feature:

```
sphinx-apidoc -f -o docs/paradrop paradrop/daemon/paradrop/
```

This scans packages in the `paradrop/daemon/paradrop` directory and creates .rst files in `docs/paradrop`. The file `reference.rst` links to `modules.rst`, connecting the newly generated code with the main documentation.

To create the documentation locally, run:

```
cd docs
make html
python -m SimpleHTTPServer 9999
```

Open your web browser of choice and point it to http://localhost:9999/_build/html/index.html.

Testing

As mentioned above, all testing is automatically run by travis-ci, a continuous integration service.

To manually run tests, install nosetest:

```
pip install nose
```

Install the required packages:

```
pip install -r docs/requirements.txt
```

Run all tests:

```
nosetests
```

How does nose detect tests? All tests live in the `tests/` directory. Nose adheres to a simple principle: anything marked with `test` in its name is most likely a test. When writing tests, make sure all functions begin with `test`.

Coverage analysis detects how much of the code is used by a test suite. If the result of the coverage is less than 100%, someone slacked. Install coveralls:

```
pip install coveralls
```

Run tests with coverage analysis:

```
nosetests --with-coverage --cover-package=paradrop
```

CHAPTER 8

Develop Applications for ParaDrop

This section of the documentation is devoted to describing the edge computing services (chutes) that run on ParaDrop. There are two categories of ParaDrop applications - pure edge applications and cloud-edge hybrid applications. The pure edge applications have standalone chutes can be deployed in the ParaDrop routers. Whereas the cloud-edge hybrid applications have both cloud part and edge part. In this section, we will focus on the chute development, in other words, the edge part. We will introduce how to develop the cloud-edge hybrid applications in <TODO>.

If this is your first time seeing ParaDrop, please start with the *Get Started* page.

Contents:

Introduction

ParaDrop is a software platform that enables services/apps to run on Wi-Fi routers. We call these apps "chutes" like a parachute.

ParaDrop runs on top of Ubuntu Core, a lightweight, transactionally updated operating system designed for deployments on embedded and IoT devices, cloud and more. It runs a new breed of super-secure, remotely upgradeable Linux app packages known as snaps. We also enable our apps through containerization by leveraging Docker.

Minimally, a chute has a Dockerfile, which contains instructions for building and preparing the application to run on ParaDrop. A chute will usually also require scripts, binaries, configuration files, and other assets. For integration with the ParaDrop toolset, we highly recommend developing a chute as a GitHub project, but other organization methods are possible.

We will examine the hello-world chute as an example of a complete ParaDrop application.

Structure

Our hello-world chute is a git project with the following files:

```
chute/index.html
Dockerfile
README.md
```

The top-level contains a README and a special file called "Dockerfile", which will be discussed below. As a convention, we place files that will be used by the running application in a subdirectory called "chute". This is not necessary but helps keep the project organized. Valid alternatives include "src" or "app".

Dockerfile

The Dockerfile contains instructions for building and preparing an application to run on ParaDrop. Here is a minimal Dockerfile for our hello-world chute:

```
FROM nginx
ADD chute/index.html /usr/share/nginx/html/index.html
```

FROM nginx

The FROM instruction specifies a base image for the chute. This could be a Linux distribution such as "ubuntu:14.04" or an standalone application such as "nginx". The image name must match an image in the Docker public registry. We recommend choosing from the official repositories. Here we use "nginx" for a light-weight web server.

ADD <source> <destination>

The ADD instruction copies a file or directory from the source repository to the chute filesystem. This is useful for installing scripts or other files required by the chute and are part of the source repository. The <source> path should be inside the respository, and the <destination> path should be an absolute path or a path inside the chute's working directory. Here we install the index.html file from our source repository to the search directory used by nginx.

Other useful commands for building chutes are RUN and CMD. For a complete reference, please visit the official Dockerfile reference.

Here is an alternative implementation of the hello-world Dockerfile that demonstrates some of the other useful instructions.

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y nginx
ADD chute/index.html /usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Here we use a RUN instruction to install nginx and a CMD instruction to set nginx as the command to run inside the chute container. Using "ubuntu:14.04" as the base image gives access to any packages that can be installed through apt-get.

Persistent Data

Each running chute has a persistent data storage that is not visible to other chutes. By default the persistent data directory is named "/data" inside the chute's filesystem. Files stored in this directory will remain when upgrading or downgrading the chute and are only removed when uninstalling the chute.

System Information

The ParaDrop daemon share system information with chutes through a read-only directory named "/paradrop". Chutes that are configured with a WiFi access point will find a file in this directory that lists wireless clients. In future versions

there will also be information about Bluetooth and other wireless devices.

dnsmasq-wifi.leases

This file lists client devices that have connected to the chute's WiFi network and received a DHCP lease. This is a plain text file with one line for each device containing the following space-separated fields.

1. DHCP lease expiration time (seconds since Unix epoch).
2. MAC address.
3. IP address.
4. Host name, if known.
5. Client ID, if known; the format of this field varies between devices.

The following example shows two devices connected to the chute's WiFi network.

```
1480650200 00:11:22:33:44:55 192.168.128.130 android-ffeeddccbbaa9988 *
1480640500 00:22:44:66:88:aa 192.168.128.170 someones-iPod 01:00:22:44:66:88:aa
```

Chute-to-Host API

The Paradrop daemon exposes some functionality and configuration options to running chutes through an HTTP API. This aspect of Paradrop is under rapid development, and new features will be added with every release. The host API is available to chutes through the URL "http://home.paradrop.org/api/v1". Paradrop automatically configure chutes to resolve "home.paradrop.org" to the ParaDrop device itself, so these requests go to the ParaDrop daemon running on the router and not to an outside server.

Authorization

In order to access the host API, chutes must pass a token with every request that proves the authenticity of the request. When chutes are installed on a ParaDrop router, they automatically receive a token through an environment variable named "PARADROP_API_TOKEN". The chute should read this environment variable and pass the token as a Bearer token in an HTTP Authorization header. Here is an example in Python using the Requests library.:

```
import os
import requests

CHUTE_NAME = os.environ.get('PARADROP_CHUTE_NAME', 'chute')
API_TOKEN = os.environ.get('PARADROP_API_TOKEN', 'NA')

headers = { 'Authorization': 'Bearer ' + API_TOKEN }
url = 'http://home.paradrop.org/api/v1/chutes/{}/networks'.format(CHUTE_NAME)
res = requests.get(url, headers=headers)
print(res.json())
```

/chutes/<chute name>/networks

- Purpose: List networks (such as Wi-Fi networks) configured for the chute.
- Methods: GET
- Returns: [object]

Note: there are currently not many different types of networks supported for chutes, so most chutes will either have no networks (empty list) or a list containing a single entry that looks like this.:

```
{ 'interface': 'wlan0', 'name': 'wifi', 'type': 'wifi' }
```

/chutes/<chute name>/networks/<network name>/stations

- Purpose: List devices ("stations") connected to a wireless network.
- Methods: GET
- Returns: [object]

For chutes that have configured a Wi-Fi AP, this endpoint provides detailed information about devices that are connected to the AP, including MAC address, bytes sent and received, and average signal strength. Here is an example response.:

```
[{'authenticated': 'yes',
  'authorized': 'yes',
  'inactive_time': '36108 ms',
  'mac_addr': '5c:59:48:7d:b9:e6',
  'mfp': 'no',
  'preamble': 'short',
  'rx_bitrate': '65.0 MBit/s MCS 7',
  'rx_bytes': '10211',
  'rx_packets': '168',
  'signal': '-42 dBm',
  'signal_avg': '-43 dBm',
  'tdls_peer': 'no',
  'tx_bitrate': '1.0 MBit/s',
  'tx_bytes': '34779',
  'tx_failed': '0',
  'tx_packets': '71',
  'tx_retries': '0',
  'wmm_wme': 'yes'}]
```

/chutes/<chute name>/networks/<network name>/stations/<mac address>

- Purpose: View or remove a device ("station") connected to a wireless network.
- Methods: GET, DELETE
- Returns: object

GET returns similar information as the request above but for a single station. DELETE will kick the device from the wireless network, but in many cases the device will be able to reconnect.

Develop a chute with ParaDrop tools

Developing Light Chutes

Light chutes build and install the same way as normal chutes and can do many of the same things. However, they make use of prebuilt base images that are optimized for different programming languages.

Light chutes offer these advantages over normal chutes.

- **Safety:** Light chutes have stronger confinement properties, so you can feel safer installing a light chute written by a third party developer.
- **Fast installation:** Light chutes use a common base image that may already be cached on the router, so installation can be very fast.
- **Simplicity:** You do not need to learn how to write and debug a Dockerfile to develop a chute. Instead, you can use the package management tools you may already be using (e.g. package.json for npm and requirements.txt for pip).
- **Portability:** With ARM suppport coming soon for ParaDrop, your light chutes will most likely run on ARM with extra work on your part. This is not the case for normal chutes that use a custom Dockerfile.

We will look at the node-hello-world chute as an example of a light chute for ParaDrop.

Structure

Our hello-world chute is a git project with the following files:

```
README.md
index.js
package.json
paradrop.yaml
```

The project contains the typical files for a node.js project as well as a special file called "paradrop.yaml".

paradrop.yaml

The paradrop.yaml file contains information that ParaDrop needs in order to run the chute. Here are the contents for the hello-world example:

```
version: 1
type: light
use: node
command: node index.js
```

All of these fields are required and very simple to use.

version: 1

This specifies the version of the paradrop.yaml schema in order to allow future changes without breaking existing chutes. You should specify version *1*.

type: light

This indicates that we are building a *light* chute.

use: node

This indicates that we are using the *node* base image for this chute. You should choose the base image appropriate for your project. Supported images are: *node* and *python2*.

command: node index.js

This line indicates the command for starting your application. You can either specified it this way, as a string with spaces between the parameters, or you can use a list of strings. The latter format would be particularly useful if your parameters include spaces. Here is an example:

```
command:
  - node
  - index.js
```

Persistent Data

Each running chute has a persistent data storage that is not visible to other chutes. By default the persistent data directory is named "/data" inside the chute's filesystem. Files stored in this directory will remain when upgrading or downgrading the chute and are only removed when uninstalling the chute.

Getting Started with Go

This tutorial will teach you how to build a "Hello, World!" chute using Go.

Prerequisites

Make sure you have Go installed as well as ParaDrop pdtools (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

Set up

Make a new directory.

```
mkdir go-hello-world
cd go-hello-world
```

Create a chute configuration

Use the pdtools interactive init command to create a paradrop.yaml file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: go-hello-world
description: Hello World chute for ParaDrop using Go.
type: light
image: go
command: app
```

The end result should be a paradrop.yaml file similar to the following.

```
command: app
config: {}
description: Hello World chute for ParaDrop using Go.
name: go-hello-world
type: light
```

```
use: go
version: 1
```

Develop the Application

Create a file name `main.go` with the following code.

```
package main

import (
    "fmt"
    "net/http"
)

func GetIndex(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World!\n")
}

func main() {
    fmt.Println("Listening on :8000")
    http.HandleFunc("/", GetIndex)
    http.ListenAndServe(":8000", nil)
}
```

Run the application locally with the following command.

```
go run main.go
```

Then load `http://localhost:8000/` in a web browser to see the result.

Wrap Up

The web server in this application listens on port 8000. We need to include that information in the paradrop.yaml file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 8000
```

Getting Started with Java

This tutorial will teach you how to build a "Hello, World!" chute using Java and Maven.

Prerequisites

Make sure you have Java 1.8+, Maven 3.0+, as well as ParaDrop pdtools (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

Set up

Use Maven to set up an empty project.

```
mvn archetype:generate -DgroupId=org.paradrop.app -DartifactId=java-hello-world -
↪DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
cd java-hello-world
```

Create a chute configuration

Use the pdtools interactive init command to create a paradrop.yaml file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: java-hello-world
description: Hello World chute for ParaDrop using Java.
type: light
image: maven
command: java -cp target/java-hello-world-1.0-SNAPSHOT.jar org.paradrop.app.App
```

The end result should be a paradrop.yaml file similar to the following.

```
command: java -cp target/java-hello-world-1.0-SNAPSHOT.jar org.paradrop.app.App
config: {}
description: Hello World chute for ParaDrop using Java.
name: java-hello-world
type: light
use: maven
version: 1
```

Develop the Application

Replace the automatically-generated application code in `src/main/java/org/paradrop/app/App.java` with the following code.

```
package org.paradrop.app;

import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class App {
    public static void main(String[] args) throws Exception {
        System.out.println("Listening on :8000");
        HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);
        server.createContext("/", new GetIndex());
        server.start();
    }
```

```
    static class GetIndex implements HttpHandler {
        @Override
        public void handle(HttpExchange t) throws IOException {
            String response = "Hello, World!";
            t.sendResponseHeaders(200, response.length());
            OutputStream os = t.getResponseBody();
            os.write(response.getBytes());
            os.close();
        }
    }
}
```

Run the application locally with the following commands.

```
mvn package
java -cp target/java-hello-world-1.0-SNAPSHOT.jar org.paradrop.app.App
```

Then load `http://localhost:8000/` in a web browser to see the result.

Wrap Up

The web server in this application listens on port 8000. We need to include that information in the paradrop.yaml file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 8000
```

Getting Started with Node.js

This tutorial will teach you how to build a "Hello, World!" chute using Node.js and Express.

Prerequisites

Make sure you have Node.js (v6 or newer) installed as well as ParaDrop pdtools (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

Set up

Make a new directory.

```
mkdir node-hello-world
cd node-hello-world
```

Create a chute configuration

Use the pdtools interactive init command to create a paradrop.yaml file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: node-hello-world
description: Hello World chute for ParaDrop using Node.js.
type: light
image: node
command: node index.js
```

The end result should be a paradrop.yaml file similar to the following.

```
command: node index.js
config: {}
description: Hello World chute for ParaDrop using Node.js.
name: node-hello-world
type: light
use: node
version: 1
```

The `pdtools chute init` command will also create a package.json file for you if one did not already exist, so there is no need to run `npm init` after running `pdtools chute init`.

Install Dependencies

Use the following command to install some dependencies. We will be using Express as a simple web server.

The `--save` option instructs npm to save the packages to the package.json file. When installing the chute, ParaDrop will read package.json to install the same versions of the packages that you used for development.:

```
npm install --save express@^4.16.1
```

Develop the Application

We indicated that index.js is the entrypoint for the application, so we will create a file named `index.js` and put our code there.

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello, World!')
})

app.listen(3000, function() {
  console.log('Listening on port 3000.')
})
```

Run the application locally with the following command.

```
node index.js
```

Then load `http://localhost:3000/` in a web browser to see the result.

Wrap Up

The web server in this application listens on port 3000. We need to include that information in the paradrop.yaml file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 3000
```

Getting Started with Python

This tutorial will teach you how to build a "Hello, World!" chute using Python and Flask.

Prerequisites

Make sure you have Python 2 installed as well as ParaDrop pdtools (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

Set up

Make a new directory.

```
mkdir python-hello-world
cd python-hello-world
```

Create a chute configuration

Use the pdtools interactive init command to create a paradrop.yaml file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: python-hello-world
description: Hello World chute for ParaDrop using Python.
type: light
image: python2
command: python2 -u main.py
```

The end result should be a paradrop.yaml file similar to the following.

```
command: python2 -u main.py
config: {}
description: Hello World chute for ParaDrop using Python.
name: python-hello-world
type: light
use: python2
version: 1
```

Install Dependencies

We will use pip and virtualenv to manage dependencies for the project. First set up a virtual enviroment.

```
virtualenv venv
source venv/bin/activate
```

Use the following command to install some dependencies. We will be using Flask as a simple web server.

```
pip install Flask==0.12.2
```

Finally, save the version information to a file called `requirements.txt`. When installing the chute, ParaDrop will use this file to install the same versions of the packages that you used during development.

```
pip freeze >requirements.txt
```

Develop the Application

We indicated that main.py is the entrypoint for the application, so we will create a file named `main.py` and put our code there.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Run the application locally with the following command.

```
python main.py
```

Then load `http://localhost:5000/` in a web browser to see the result.

Wrap Up

The web server in this application listens on port 5000. We need to include that information in the paradrop.yaml file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 5000
```

Chute Deployment

Starting with version 0.2, developers should use our website, paradrop.org for creating chutes and installing them on Paradrop routers. The command line tools from Paradrop 0.1 are not currently supported.

As our website is under development, please refer to the instructions on the website or contact us for help with making and installing a chute.

Tutorial: Sticky Board

This tutorial will teach you how to build a fully-functional ParaDrop application from scratch. Through the tutorial, we will build a "Sticky Board", a local board where visitors can post images for others to see. We will be using Node.js to build the application, so make sure you have that installed on your development machine.

Set Up

Make a new directory, and initialize a git repository:

```
mkdir sticky_board
cd sticky_board
git init
mkdir views
```

Setup Node.js Project

We will be using npm to manage Node.js packages. You can use the `npm init` command to get started or create a file called package.json. with the following contents:

```
{
  "name": "sticky_board",
  "version": "1.0.0",
  "description": "Post images for others to see.",
  "main": "index.js",
  "author": "ParaDrop Team"
}
```

Install Dependencies

Use the following command to install some dependencies that we will be using to build the application. We use express as a simple web server along with a plugin for accepting file uploads. We will also use Embedded JS (EJS) for simple templating, demonstrated later in this tutorial.

The `--save` option instructs npm to save the packages to the package.json file. ParaDrop will read package.json to install the same versions of the packages that you used for development.

```
npm install --save ejs@^2.5.6 express@^4.14.1 express-fileupload@^0.1.1
```

Hello World

Let's start with a minimal Hello World Express.js example. Create a file named `index.js` and add the following code:

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});
```

```
app.listen(3000, function() {
  console.log('Listening on port 3000.');
});
```

Run the app with the following command:

```
node index.js
```

Then load `http://localhost:3000/` in a web browser to see the result.

Image Uploads

Next, we will add an endpoint to receive image uploads.

```
var express = require('express');
var fileupload = require('express-fileupload');

var app = express();

// Use PARADROP_DATA_DIR when running on Paradrop and /tmp for testing.
var storage_dir = process.env.PARADROP_DATA_DIR || '/tmp';

app.use(fileupload());
app.use(express.static(storage_dir));
app.set('view engine', 'ejs');

app.post('/create', function(req, res) {
  var img = req.files.img;
  if (img) {
    img.mv(storage_dir + '/' + img.name);
  }

  res.redirect('/');
});

app.get('/', function (req, res) {
  res.render('home');
});

app.listen(3000, function() {
  console.log('Listening on port 3000.');
});
```

Create a new file in the views directory called home.ejs with the following contents:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>ParaDrop Sticky Board</title>
  </head>
  <body>
    <h1>ParaDrop Sticky Board</h1>
    <h2>Create a Note</h2>
    <p>Upload an image file to create a note for others to see.</p>
    <form action="/create" method="POST" encType="multipart/form-data">
      <input type="file" name="img" />
```

```
      <input type="submit" value="Create" />
    </form>
  </body>
</html>
```

Right now it is just plain HTML. In the next section we will make use of templating to add images to the sticky board.

Run the app again and load `http://localhost:3000/`. Try using the form to upload an image. You should then be able to find your image by loading `http://localhost:3000/<filename>`.

Displaying Notes

The last thing the app needs to be able to do is display all of the notes that people have posted. First, add some logic to index.js to keep track of the most recent image uploads:

```
var express = require('express');
var fileupload = require('express-fileupload');

var app = express();

// Use PARADROP_DATA_DIR when running on Paradrop and /tmp for testing.
var storage_dir = process.env.PARADROP_DATA_DIR || '/tmp';

// Maximum number of notes to display.
var max_visible_notes = process.env.MAX_VISIBLE_NOTES || 16;

app.locals.notes = [];
for (var i = 0; i < max_visible_notes; i++) {
  if (i % 2 == 0) {
    addNote('http://pages.cs.wisc.edu/~hartung/paradrop/paradrop.png');
  } else {
    addNote('http://pages.cs.wisc.edu/~hartung/paradrop/paradrop_inverted.png');
  }
}

function addNote(img) {
  app.locals.notes.push({
    img: img,
  });

  if (app.locals.notes.length > max_visible_notes) {
    app.locals.notes = app.locals.notes.slice(-max_visible_notes);
  }
}

app.use(fileupload());
app.use(express.static(storage_dir));
app.set('view engine', 'ejs');

app.post('/create', function(req, res) {
  var img = req.files.img;
  if (img) {
    img.mv(storage_dir + '/' + img.name);
    addNote(img.name);
  }

  res.redirect('/');
```

```
});

app.get('/', function (req, res) {
  res.render('home');
});

app.listen(3000, function() {
  console.log('Listening on port 3000.');
});
```

The paradrop.png and paradrop_inverted.png are just used as fillers until people post other images. Feel free to use different images.

Also, update home.ejs:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>ParaDrop Sticky Board</title>

    <style>
      div.holder {
        float: left;
        min-width: 240px;
        width: 24%;
        padding: 5px 5px;
      }

      div.separator {
        clear: both;
      }
    </style>
  </head>
  <body>
    <h1>ParaDrop Sticky Board</h1>

    <div>
      <% for(var i = 0; i<notes.length; i++) {%>
        <div class="holder">
          <img src="<%= notes[i].img %>" width="100%"></img>
        </div>
      <% } %>
    </div>

    <div class="separator"></div>

    <h2>Create a Note</h2>
    <p>Upload an image file to create a note for others to see.</p>
    <form action="/create" method="POST" encType="multipart/form-data">
      <input type="file" name="img" />
      <input type="submit" value="Create" />
    </form>
  </body>
</html>
```

We use some Embedded JS code to loop over the array of notes stored in `app.locals.notes` and generate an img element for each one with the appropriate filename.

Now when you run the app and load `http://localhost:3000/` you should see the filler images. Try using the form to upload an image, and it should appear on the board.

Preparing the Chute

Create a file called paradrop.yaml with the following contents:

```
version: 1
type: light
use: node
command: node index.js
```

This file tells ParaDrop a few things about how to run your code on a ParaDrop gateway.

Finally, add all of your new files to the git repository:

```
git add index.js package.json paradrop.yaml views/home.ejs
git commit -m "Created sticky board from tutorial"
```

Create a new repository on github.com and follow their instructions to push your code to github.

Registering the Chute with ParaDrop

Log on to paradrop.org and go to the Chute Store tab. Click "Create Chute" and give your chute a name and description. You may need to be creative with the name because the chute store requires unique names. Then click "Submit".

Next, click "Create Version". For this tutorial, there are only two important fields to fill out on this form. First, check the box to "enable web service" and enter the number 3000 because that is the port we chose in index.js. Second, select "Download from URL" for Project source and enter the github URL for your project. Then click "Submit".

Congratulations! You have made a ParaDrop chute. If you have a ParaDrop router, you should now be able to install the chute on your router. If not, you can follow the Getting Started guide to set up a VM running ParaDrop.

CHAPTER 9

Known Issues

Please check here for issues during setup.

Issues with the hardware or operating system

Issue 1: Docker fails to start after a reboot

This can happen if the 'docker.pid' file was not properly cleaned up, which causes the docker daemon to conclude that it is already running.

To fix this, remove the pid file on the router and reboot.

```
sudo rm /var/lib/apps/docker/1.11.2/run/docker.pid
sudo reboot
```

Issue 2: WiFi devices are not detected after a reboot

Occasionally, when routers start up the WiFi devices are not detected properly. When this happens the command `iw dev` will display nothing instead of the expected devices. This is usually remedied by rebooting.

CHAPTER 10

Frequently Asked Questions

Please check here for any FAQ's.

CHAPTER 11

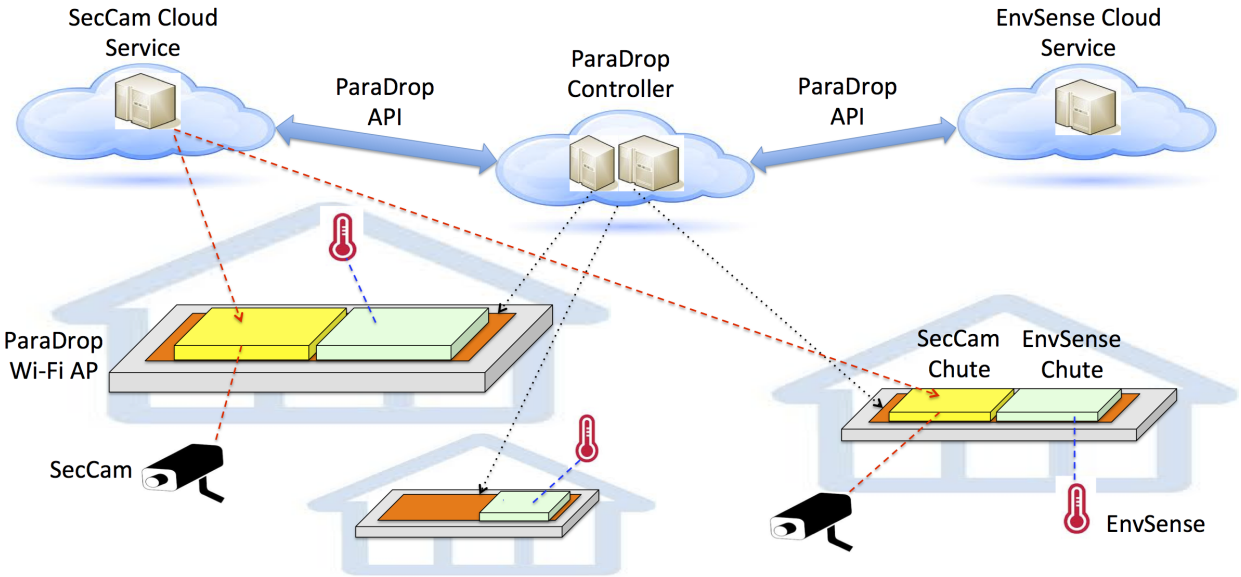
ParaDrop package references

paradrop/modules

CHAPTER 12

ParaDrop - Enabling Edge Computing at the Extreme Edge

ParaDrop is an open source edge computing platform developed by the WiNGS Lab at the University of Wisconsin-Madison. We built the ParaDrop platform with WiFi routers, so that we can "paradrop" services from the cloud to the extreme wireless edge - just one hop from user's mobile devices, data sources, and actuators of IoT applications. The name "ParaDrop" comes from the ability to "drop" supplies and resources ("services") into the network edge.



The above figure gives a high level overview of ParaDrop, including the ParaDrop platform and two example applications. With the ParaDrop API, third-party applications can deploy services into the network edge - the WiFi routers. More information about the design and evolution of ParaDrop can be found in the paper.

CHAPTER 13

Getting Started

Please visit the *Get Started* page for a quick introduction about how to use ParaDrop.

CHAPTER 14

Where to go from here?

We have document about ParaDrop application development found under *Develop Applications for ParaDrop*. If you are interested in working on the development of the ParaDrop platform (our github code) then check out: *How to Contribute*.