paradrop Documentation

Release 0.5

Paradrop Labs

Apr 21, 2017

Contents

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.

Contents:

CHAPTER 1

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/0.5/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.

Activating your Router

Follow these steps the first time you start a new physical or virtual Paradrop router. Activation associates the router with your account on paradrop.org so that you can manage the router and install chutes from the chute store.

1. Make an account on paradrop.org if you do not have one.
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://localhost:8080. If you are prompted to login, the default username is *paradrop* with an empty password. Click the "Activate the router witha 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.

Installing Chutes

1. Make an account on paradrop.org and 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://localhost:8000. Once the installation is complete, test it in a web browser.

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.

CHAPTER 2

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/0.5/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
```

Continue to *Activating a Router*.

CHAPTER 3

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
...
```

Continue to *Activating a Router*.

CHAPTER 4

Activating a Router

Follow these steps the first time you start a new physical or virtual ParaDrop router. Activation associates the router with your account on paradrop.org so that you can manage the router and install chutes from the chute store.

1. Make an account on paradrop.org if you do not have one.
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_address>. 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.

CHAPTER 5

Apps on Paradrop

This section of the documentation is devoted to describing the apps that run on Paradrop.

If this is your first time seeing Paradrop, please start with the gettingstarted page.

Contents:

Chute Development

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 chute/index.html index.html

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 instance tools 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 WiFi AP 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 Paradrop, 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.

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, *heavy* 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.

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 6

The Paradrop Instance System

This section focuses on the *Instance Tools*. This is the set of daemons and tools required to allow the Paradrop platform to function on virtual machines and real hardware. Use the information below to learn about Snappy Ubuntu and how we leverage it to create next generation smart routers.

Contents:

Build System

Paradrop includes a set of build tools to make development as easy as possible.

Currently this system takes the form of a bash script that automates installation and execution, but in time this may evolve into a published python package. This page outlines the steps required to manually build the components required to develop with paradrop.

Components in the build process:

- *Installing and running Ubuntu Snappy*
- *Building paradrop*
- *Installing paradrop*
- **'Creating chutes'_**

We recommend using Ubuntu 14.04 as the build environment for this version of Paradrop. Ubuntu 16.04 will not work because the snappy development tools have changed. The next release of Paradrop will use the new tools.

You will only need to follow these instructions if you will be making changes to the Paradrop instance tools. Otherwise, you can use our pre-built Paradrop disk image. Please visit the ../chutes/gettingstarted page.

Installing and running Ubuntu Snappy

Snappy is an Ubuntu release focusing on low-overhead for a large set of platforms. These instructions are for getting a Snappy instance up and running using 'kvm'.

Download and unzip a snappy image:

```
wget http://releases.ubuntu.com/15.04/ubuntu-15.04-snappy-amd64-generic.img.xz
unxz ubuntu-15.04-snappy-amd64-generic.img.xz
```

Launch the snappy image using kvm:

```
kvm -m 512 -redir :8090::80 -redir :8022::22 ubuntu-15.04-snappy-amd64-generic.img
```

Connect to local instance using ssh:

```
ssh -p 8022 ubuntu@localhost
```

Building paradrop

Snappy is a closed system (by design!). Arbitrary program installation is not allowed, so to allow paradrop access to the wide world of `pypi` the build system relies on two tools.

- `virtualenv` is a tool that creates encapsulated environments in which python packages can be installed.
- `pex` can compress python packages into a zip file that can be executed by any python interpreter.
- `snappy` is a tool for building snap packages. *Note:* Ubuntu 16.04 uses snapcraft instead, which produces incompatible packages.

First, set *DEV_MACHINE_IP=paradrop.org* in pdbuild.conf. The build script will refuse to run if this variable is not set.

Install the necessary development tools:

```
./pdbuild.sh setup
```

Build the Paradrop snap package:

```
./pdbuild.sh build
```

Installing paradrop

Install dependencies on the virtual machine:

```
./pdbuild.sh install_deps
```

Install the newly created Paradrop snap package:

```
./pdbuild.sh install_dev
```

Snappy Confinement

Snappy confines running applications in two ways: directory isolation and mandatory access control (MAC). Directory isolation means the application cannot leave its installed directory. MAC means the application cannot execute any system commands or access any files it does not have explicit, predetermined permissions to.

MAC is the more serious hurdle for paradrop development. Snaps declare permissions through an AppArmor profile.

Getting started with Profile Generation

Install tools and profiles:

```
sudo apt-get install apparmor-profiles apparmor-utils
```

List active profiles:

```
sudo apparmor_status
```

Profiles in complain mode log behavior, while those in enforce mode actively restrict it.

The following steps assume paradrop is installed on the system and not on a virtualenv.

Create a new, blank profile:

```
cd /etc/apparmor.d/
sudo aa-autodep paradrop
```

Use aa-complain to put the profile in complain mode:

```
sudo aa-complain paradrop
```

Excercise the application! AppArmor will surreptitiously watch the program in the background and log all behavior. Once finished, use the following command to go through the resulting requests, approve or deny them, and autogenerate a profile:

```
sudo aa-logprof
```

Documentation and tests

Documentation is handled by sphinx and readthedocs.

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

Documentation

Information about docs creation, management, and display.

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 `src/` and build documentation from the docstrings within.

For example, the directive `.. automodule:: paradrop.backend` will build all the documentation for the given package. See google for more instructions.

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/api paradrop/paradrop/
```

This scans packages in the `src/paradrop` directory and creates .rst files in `docs/api`. The root file `index.rst` links to `modules.rst`, connecting the newly generated api 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
```

Well thats easy. 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 7

Architecture

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

This is a work in progress. Check back later!

CHAPTER 8

Known Issues

Please check here for issues during setup.

Issues using pdbuild.sh

These issues are related to the *Instance Tools* found on github.

Issue 1: `pdbuild.sh install` fails

Docker snap is missing inside of virtual router, run `pdbuild.sh install_deps`:

```
issues while running ssh command: Installing /tmp/paradrop_0.1.0_all.snap
2015/08/11 21:22:57 Signature check failed, but installing anyway as requested
/tmp/paradrop_0.1.0_all.snap failed to install: missing frameworks: docker
```

Issue 2: `pdbuild.sh install` fails

This is a known issue for the Paradrop team, if you get this please email us at developers@paradrop.io:

```
Installing paradrop_0.1.0_all.snap from local environment

issues while running ssh command: Installing /tmp/paradrop_0.1.0_all.snap
2015/08/11 21:29:48 Signature check failed, but installing anyway as requested
/tmp/paradrop_0.1.0_all.snap failed to install: [start paradrop_pdconfd_0.1.0.service]
failed with exit status 1: Job for paradrop_pdconfd_0.1.0.service failed.
See "systemctl status paradrop_pdconfd_0.1.0.service" and "journalctl -xe" for␣
↪details.
```

Issue 3: `pdbuild.sh up` fails

This is very common and will happen if you delete your VM and setup a fresh one, which will have a different key. The solution is simple and is stated in the error message. Follow the instructions to remove the key from your known_hosts file.

```
Failed to setup keys: failed to setup keys: issues while running ssh command:
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
e6:ec:b1:93:7d:91:84:50:19:36:14:8e:ce:ef:6a:0b.
Please contact your system administrator.
```

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 9

Frequently Asked Questions

Please check here for any FAQ's.

CHAPTER 10

Paradrop

Paradrop is a software platform that enables apps to run on Wi-Fi routers. We call these apps “chutes” like a parachute. The name Paradrop comes from the fact that we are enabling the ability to “drop” supplies and resources (“apps”) into a difficult and not well-travelled environment - the home.

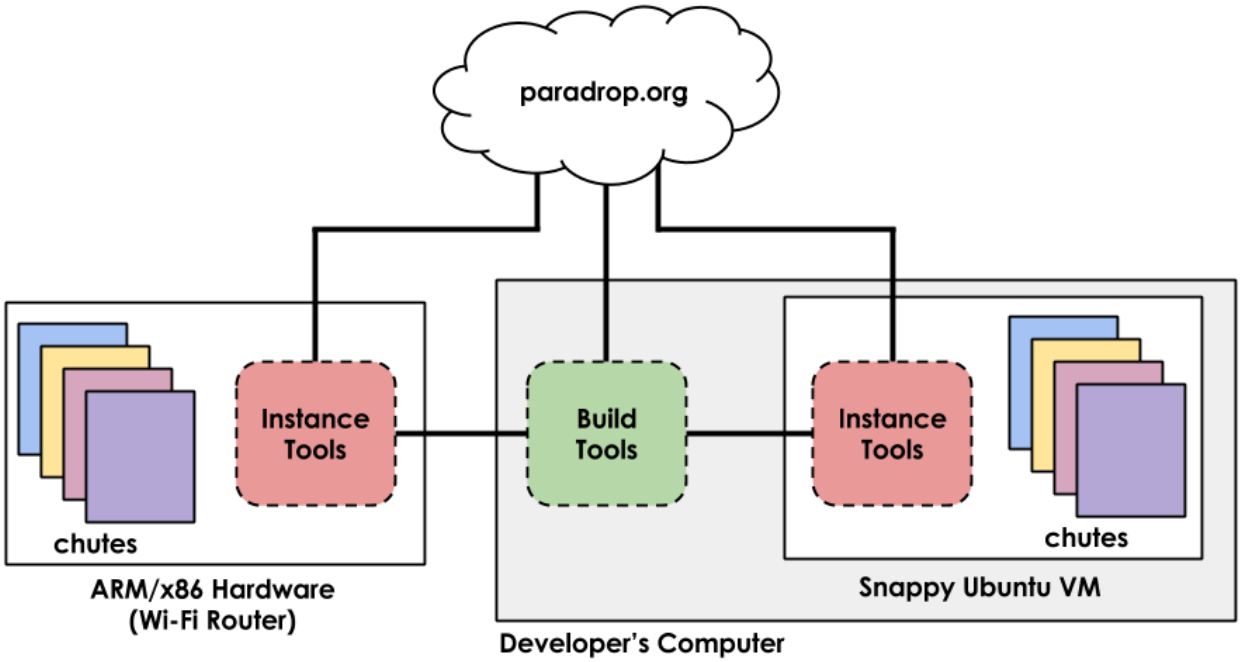
Paradrop runs on top of Snappy Ubuntu, a trimmed-down and secured operating system that can run on ARM and x86. We also enable our apps through containerization by leveraging Docker.

CHAPTER 11

The Paradrop workflow

There are two components to the Paradrop platform:

- The build tools - our CLI that enables registration, login, and control. With version 0.2 and up, Paradrop routers can be managed through our cloud management service instead of the CLI.
- The instance tools - our configuration daemons and tools to launch apps on hardware.



As you can see from the image above, we will refer to *Build Tools* when we talk about the CLI program running on your development computer that controls and communicates with the rest of the Paradrop platform. Treat this tool as your window into the rest of the Paradrop world. Our *Instance Tools* leverage programs like Docker to allow Paradrop apps to run on router hardware. This "hardware" could be a Raspberry Pi, or even a virtual machine on your computer

that acts as a router (which is why we call it an Instance sometimes). Using Paradrop, you can actually plug in a USB Wi-Fi adapter and turn a virtual machine on your computer into a real router with our platform!

CHAPTER 12

Getting Started

Please visit the chutes/gettingstarted page for a quick introduction to Paradrop.

CHAPTER 13

Where to go from here?

We have advanced app examples found under *Apps on Paradrop*. If you are interested in working on the instance side of paradrop (our github code) than check out: *The Paradrop Instance System*.

CHAPTER 14

What if I don't have Ubuntu?

With version 0.2 and up, all of Paradrop's capabilities can be managed through our web-based service at paradrop.org.