Marketplace Documentation

Release 0.1

Marketplace Developers

August 03, 2016

Contents

High-level developer documentation for the Firefox Marketplace.

CHAPTER 1

Related Documentation

- Firefox Marketplace API documentation
- Firefox Marketplace frontend documentation
- Internal Mozilla documentation

CHAPTER 2

Contents

2.1 Contributing

We love having contributors for Firefox Marketplace! To contribute to the Marketplace:

- Find a bug to work on. This is a list of bugs we think are good for contributors, but you are certainly welcome to work on any Marketplace bug.
- Assign the bug to yourself.
- Set up your environment.
- Is your bug something on the frontend (such as the consumer-facing Marketplace)? Then set up the *Marketplace Frontend*.
- Is your bug something on the backend, payments, developer pages, or reviewer tools? Then set up the *Marketplace Backend*.
- Work on your bug. Feel free to reach out to over Marketplace developers on IRC in the #marketplace channel, mailing lists, or on the bug itself.
- Write unit tests if appropriate.
- For code changes, ensure that you follow the Mozilla WebDev style guide.
- For visual changes, ensure that you follow the Maketplace Design style guide.
- Submit a pull request.
- A reviewer will review your code and either give an approval (shown as **r+**) or not. Follow up on any comments and improve the pull request. When approved, the reviewer will merge.
- Update the bug, pasting the URL of the commit into the bug, setting the milestone to the closest date in the future and close the bug.

That's it, thanks!

2.2 Overview

The Firefox Marketplace is a collection of services and repositories that together form the Marketplace.

Our recommended setup for the backend is to use *Docker*. And our recommended setup for the frontend is to clone the repository as detailed in our *Marketplace Frontend* documentation. By default, the frontend setup will point to our development server and database. But if you wanted a full environment, you can set up both the backend and frontend, and point your frontend towards the backend set up by Docker.

2.2.1 Setting up the Backend

If you want to work on the backend, the API, or the payments infrastructure, visit our *Marketplace Backend* documentation. It will explain how to set up Docker as it automates the setup of the Marketplace environment.

2.2.2 Setting up the Frontend

If you want to work on the consumer-facing Marketplace frontend, visit our *Marketplace Frontend* documentation. It only takes three commands to get things up and running. You don't even need to setup a backend, although you could if you wanted your own personal playground.

2.2.3 List of Services and Repositories

Backend

- **Zamboni**: the main API backend that also serves developer, reviewer, and admin tool pages. Written in Python with Django - source, docs, API docs.
- **Marketplace API Mock**: a fake API backend used for frontend testing. Written in Python with Flask - source.
- **Monolith**: storage and query server for statistics around the Marketplace. Written in Python - source, aggregator source.
- **Webpay** and **Solitude**: servers for processing payments for the Marketplace. Written in Python with Django - webpay source, webpay docs, solitude source, solitude docs.
- **Trunion**: a signing service for app receipts and packaged apps. Written in Python with Pyramid - source.
- **Zippy**: a fake backend for payments so to fake out carrier billing. Written in Node - source.
- **APK Factory**: generates Android APKs out of web apps. Written in Node and Python - factory source, signer source, signer docs.
- **marketplace-env**: automated Marketplace backend setup using Docker. Uses Docker and fig - source
- **Signing server**: signs receipts and apps for the Marketplace. Written in Python - signing service, and signing libraries.

Frontend (Javascript)

- **Fireplace**: the main frontend for the Firefox Marketplace. Written in Javascript with our Commonplace framework - source.
- **Statistics**: dashboard that displays charts and graphs from Monolith. Written in Javascript with our Commonplace framework - source.
- **Transonic**: curation and editorial tools for the Marketplace, notably for the Feed. Written in Javascript with our Commonplace framework - source.
- **Operator Dashboard**: dashboard for FirefoxOS operators to manage their app collections. Written in Javascript with our Commonplace framework - source.
- **Commbadge**: dashboard for communications between app reviewers and app developers. Written in Javascript with our Commonplace framework - source.
- **Spartacus**: the frontend for Webpay. Written in Javascript - source.

Frontend Components (Javascript)

- **marketplace-core-modules**: core JS modules for Marketplace frontend projects Written in Javascript - source.
- **commonplace**: Node module that includes configuration, template optimization, l10n. Written in Node - source.
- **marketplace-gulp**: gulpfiles for Marketplace frontend projects for builds. Written in Node - source.
- **marketplace-constants**: shared constants between the backend and frontend. Written in Python - source.
- **marketplace-elements**: web component UI elements Written in Javascript - source.

2.2.4 Marketplace Servers

Marketplace's servers are outlined below For specifics of our production configuration, please see the Services documentation (only visible to Mozilla employees).

- dev: updated each commit.
- alt dev: updated each commit. Used for testing disruptive development features.
- stage: updated when tags are made. This is as similar to production as possible. Used for testing features before they go to production. Uses real money for payments.
- payments alt: updated each commit. Used for testing dispruptive payments features. Uses real money for payments.
- production: updated as per the push schedule. The production servers are the only ones with any uptime expectations.

2.3 Marketplace Backend

Note: Upgrading to boot2docker/docker 1.3+ is required for built-in shared folder support on OSX.

Using Docker is the recommended configuration for developers to set up the Marketplace backend and payments. Docker automates a lot of the steps on setting up the backend.

It will also set up the Marketplace frontend for you to poke around, but Docker is currently not recommended to use (besides as a backend) for frontend development. If you are more interested in developing the frontend, visit our *Marketplace Frontend* docs.

For further information about the Marketplace backend, check out our API documentation and Zamboni documenation.

2.3.1 Requirements

- Mac OS X or Linux.
- A github account (set in step 3).
- packages: python2.7, python-pip, curl, python-dev

2.3.2 1. Install Docker

Install Docker.

On OSX

On Mac the docker libs can be installed using Mac Homebrew:

```
brew install caskroom/cask/brew-cask
brew cask install virtualbox
brew install docker
brew install docker-machine
brew install docker-compose
```

On Linux

The generic (check docker link above for distribution specific) install script for docker is:

```
curl -sSL https://get.docker.com/ | sh
```

Ubuntu users probably want to add ubuntu to the docker group (you need to log in and out again after):

```
sudo usermod -aG docker ubuntu
```

To install docker-compose:

```
sudo pip install docker-compose
```

2.3.3 2. Build dependencies

OSX

Let's start by creating the virtual machine that our containers will be created in:

```
docker-machine create --driver virtualbox --virtualbox-memory 2048 mkt
```

This creates a virtual machine using virtualbox with 2GB of RAM available named 'mkt'.

Next, execute the following command to export the environment variables:

```
eval "$(docker-machine env mkt)"
```

OSX and Linux

Next we will checkout the code repositories needed for development. Check out the following repositories somewhere on your machine under the same root directory (e.g.: ~/sandbox/):

- fireplace
- solitude
- solitude-auth
- spartacus
- webpay
- zamboni
- zippy

Get the marketplace environment repo and set up the configuration files needed for docker-compose. Still in the sandbox directory:

```
git clone https://github.com/mozilla/marketplace-env.git
cd marketplace-env
python link-sources.py --root $(dirname $(pwd))
```

Set up the environment variable that *docker-compose* looks for:

```
export COMPOSE_FILE=`pwd`/docker-compose.yml
export COMPOSE_PROJECT_NAME=mkt  # String prepended to every container.
```

It is recommended that you add these environment variables (the ones above, and for Mac OS those provided by `docker-machine env mkt` also) in your profile so that they persist and are consistent.

Next add a hosts entry for mp.dev.

On OSX:

```
sudo sh -c "echo $(docker-machine ip mkt 2>/dev/null)  mp.dev >> /etc/hosts"
```

On Linux:

```
sudo sh -c "echo 127.0.0.1  mp.dev >> /etc/hosts"
```

2.3.4 3. Build and run

Now that the marketplace github repos are cloned and linked we can start creating the virtual machine and docker containers.

Let's pull down the docker images and build the containers:

```
docker-compose pull
```

Note: This can take a long time the first time.

Next, start the containers:

```
docker-compose up -d
```

Note: On first run this may take a few minutes as it sets up the services, creates data, and populates the search index.

When everything is running open up a browser to http://mp.dev

2.3.5 4. Shutting down and restarting

On the Marketplace team we have found it good practice to shut down docker at the end of each work day.

OSX

To do so you can run the following commands:

```
docker-compose stop
docker-machine stop mkt
```

To start up again simply do:

```
docker-machine start mkt
docker-compose up -d
```

Linux

To do so you can run the following commands:

```
docker-compose stop
```

To start up again simply do:

```
docker-compose start -d
```

2.3.6 Issues

Come talk to us on irc://irc.mozilla.org/marketplace if you have questions, issues, or compliments.

2.4 Marketplace Backend Details

2.4.1 *docker-compose* Commands

This is not a full list of commands, for that see the *docker-compose docs <https://docs.docker.com/compose/>*, just notable ones.

`docker-compose up [-d]`

Use this if you would like all the service logs in the foreground run:

```
docker-compose up  # Ctrl-c here will shutdown all services.
```

`docker-compose logs`

View log output from a project's container.

2.4.2 *docker* Commands

For docs see: https://docs.docker.com/

FAQ

2.4.3 How do I run commands?

It's easiest to think of the containers as Linux machines you can shell into to run various commands.

Docker labels each container with a prefix of 'mkt' (as defined by `COMPOSE_PROJECT_NAME` environment variable) and a numeric suffix, which will be 1 unless you are running multiple of the same container.

So you can shell into any container by running (for zamboni, e.g.):

```
docker exec -ti mkt_zamboni_1 /bin/bash
```

From there you can run migrations:

```
schematic migrations/
```

or the unit tests:

```
./manage.py test --noinput -s --logging-clear-handlers
```

Another option is to spawn up a new instance of the container temporarily to run a command, e.g.:

```
docker-compose run --rm zamboni schematic migrations/
```

2.4.4 How do I update python/node package deps (rebuild the container)?

1. First stop the machine with your OS-specific instructions.
2. Then pull the newest images from Docker Hub:

   docker-compose pull [project]
3. Then start the machine with your OS-specific instructions.

2.4.5 How do I add an admin in Zamboni with docker?

Simply run this command replacing name@email.com with the email of the user you've recently logged-in as:

```
docker-compose run --rm zamboni python manage.py addusertogroup name@email.com 1
```

2.4.6 How do I upgrade docker?

Upgrade a machine to the latest version of Docker:

```
docker-machine upgrade
```

Environment

To configure the services in the Marketplace, you can either override each project's settings file (see documentation on each project for how that would look). Or you can alter a few environment variables that all the projects use. This is the **recommended approach** for setting up the Marketplace until you feel more comfortable with the settings in the Marketplace.

This documentation assumes that you know how to set environment variables on your development platform.

Environment variable	Used by	Description	Default
	MARKET-PLACE_URL		Webpay
	URL to nginx		http://localhost/
MEM-CACHE_URL	Zamboni, Webpay, Solitude	The location of memcache	localhost:11211
REDIS_URL	Zamboni	URL to redis	redis://localhost:6379
SOLI-TUDE_DATABASE	Solitude	dj_database_url compliant URL to solitude Mysql	mysql://root@localhost:3306/solitude
SOLITUDE_URL	Zamboni, Webpay	URL to solitude instance	http://localhost:2602
SPARTA-CUS_STATIC	Webpay	URL to Spartacus static files	http://localhost:2604
ZAM-BONI_DATABASE	Zamboni	dj_database_url compliant URL to zamboni Mysql	mysql://root@localhost:3306/zamboni
RABBIT_HOST	Rabbit	Rabbit hostname	localhost

2.4.7 Other Environment Variables

Please be aware that other parts of the site infrastructure can be affected by environment variables. Some examples:

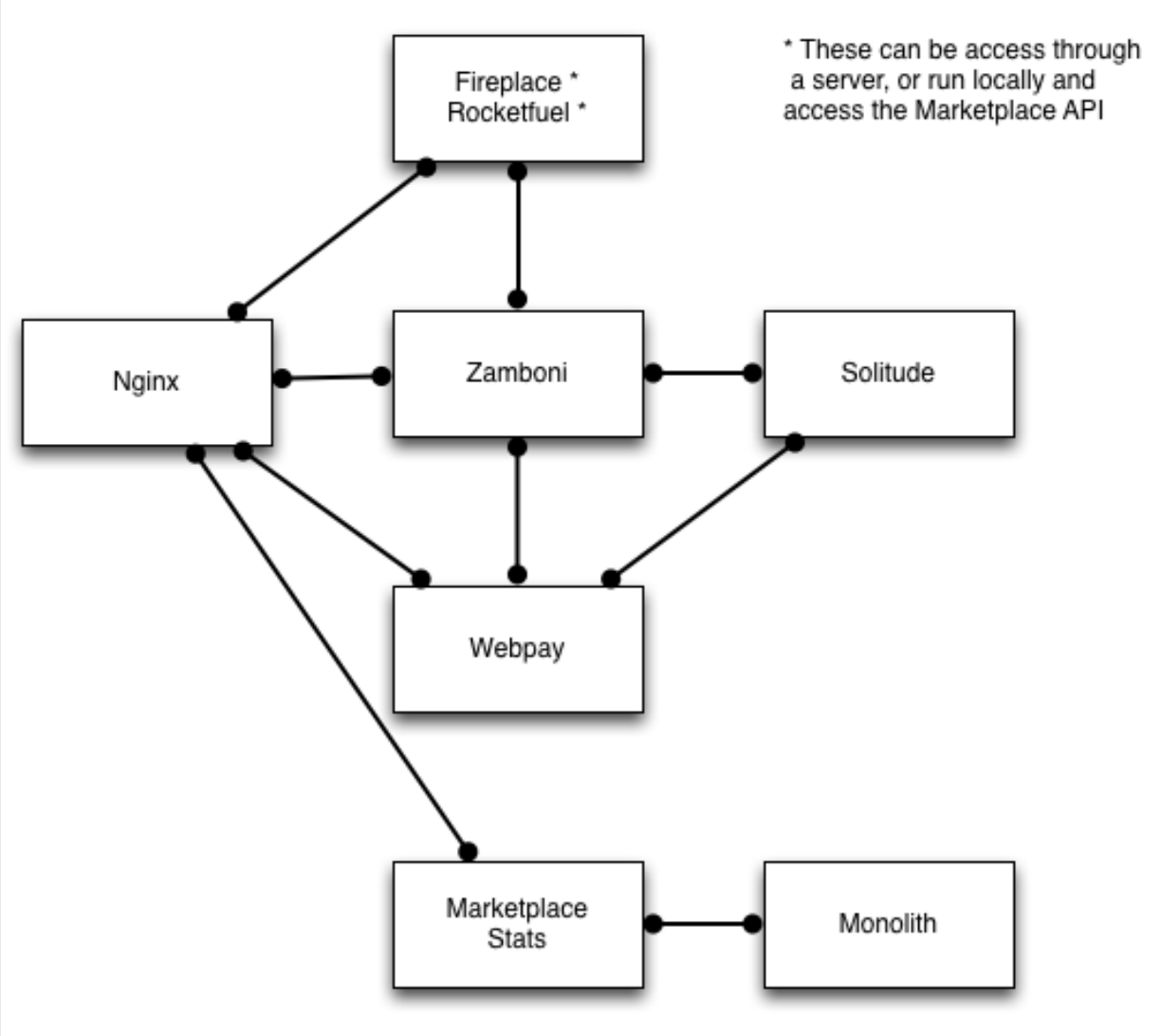
- If you want to use custom Django settings, you can set DJANGO_SETTINGS_MODULE

2.4.8 Serving With Nginx

Marketplace is designed to be an app accessible at one domain, hitting Nginx.

Behind the scenes Nginx will proxy to the other servers on your behalf.

Most developers are using Nginx to serve out the multiple services. Your configuration may look something like this:



You can find an example configuration file in our Docker repository.

2.4.9 Default Ports

By default, the services listen to the following ports:

Project	Port
Zamboni	2600
Webpay	2601
Solitude	2602
Solitude Proxy	2603
Spartacus	2604
Zippy	2605
Signing server	2606
Fireplace	8675
Commbadge	8676
Statistics	8677
Transonic	8678
Operator Dashboard	8679
Receipt verifier	9000

2.4.10 External services

The Marketplace interacts with multiple remote services that are not under the control of the Marketplace team.

Marketplace server	Reason	External	Type
marketplace-dev.allizom.org	Payments	Zippy	Test
	Content Ratings	IARC	Test
	Authentication	Firefox Accounts	Latest
marketplace.allizom.org	Payments	Bango	Prod
	Payments	Boku	Prod
	Content Ratings	IARC	Test
	Authentication	Firefox Accounts	Prod
payments-alt.allizom.org	Payments	Bango	Prod
	Payments	Boku	Prod
	Content Ratings	IARC	Test
	Authentication	Firefox Accounts	Latest
marketplace.firefox.com	Payments	Bango	Prod
	Payments	Boku	Prod
	Content Ratings	IARC	Prod
	Authentication	Firefox Accounts	Prod

Notes:

- **Zippy**: is a reference implemention of the Marketplace Payments Specification to enable easy testing and development.
- **Bango and Boku**: do not provide test instances.
- **Boku**: uses a different set of integrator keys for different servers, please see the internal docs on mana.
- **Firefox Accounts**: native flow on a device connects to the production Firefox Accounts. The web based flow connects to the servers as noted above.

2.5 Marketplace Frontend

Note: Check out our complete Marketplace frontend documentation page at marketplace-frontend.readthedocs.org.

This section is briefly about setting up the Marketplace frontend. For further information about Marketplace frontend projects, check out our dedicated frontend documentation.

2.5.1 Setup

Setting up an instance of the Marketplace frontend for development is very simple. Note, you will need Node and NPM set up on your machine:

```
git clone git@github.com:mozilla/fireplace
npm install
make install
make serve
```

This will clone the project, install dependencies, copy assets from Bower into the project, create a require.js configuration with paths set up, and start a local webserver on port 8675 by default.

2.5.2 Firefox OS

To develop on Firefox OS, we recommend using Firefox's WebIDE. Follow the instructions on connecting a device.

To generate a package (i.e., a Marketplace app to install on the device):

```
make package
```

Then install the package onto the device using WebIDE.

2.5.3 Android

To develop on Android, you'll need to make sure that the Android device can access your local development instance.

If you're running your development environment on a vm then being able to accessing it from an android device can be a challenge. To solve this problem on android there's a couple of options.

- Use a proxy with DNS spoofing like Charles proxy http://www.charlesproxy.com/. Using a recent android phone you can configure a proxy under the advanced settings of a wi-fi connection. This doesn't require root.
- Root your phone and push a hosts file to it with adb.

2.6 Marketplace Payments

This section is about processing payments on the Marketplace.

2.6.1 Setup

To develop payments locally we would recommend that you install Webpay, Solitude, Zippy and Spartacus. This will give you a complete end to end flow of how payments would work but you will be missing:

- actual carrier or credit card charges
- actual carrier interaction, such as user identification

If you want that then you'll need to set up integration with a real payment provider, such as Bango. That is discussed in the solitude docs.

The following *assumes* that you have a successfully working Spartacus, Webpay, Solitude, Zippy and Zamboni installed. This document discusses how you need to configure them all to interact.

If you are unsure on some of these settings:

- Most other developers have these settings done and are happy to help
- Each project's settings files are in GitHub with examples and documentation

Webpay and Zamboni

There is one setting that is not configured correctly out of the box at this time. You must get OAuth keys for a user in the Marketplace and then create a local settings file that overrides the following:

- `MARKETPLACE_OAUTH`: a dictionary with two keys, `key` and `secret` which point to an API key on zamboni. The user identified by the key must have the following permissions on the Marketplace: `Transaction:NotifyFailure` and `ProductIcon:Create`.

For example:

```
from base import *

MARKETPLACE_OATH = {'key': 'a', 'secret': 'b'}
```

Once you've added these in, you can test this works by hitting the URL http://your.local.webpay/mozpay/services/monitor. The value for `marketplace` should be `ok`. For example: https://marketplace.allizom.org/mozpay/services/monitor.

Webpay and Spartacus

Webpay serves views which the 'Single Page App' Spartacus renders. As a result of this webpay has several settings which control how Spartacus is served.

Webpay assumes that Spartacus is running at http://localhost:2604 by default. To run Spartacus you should move into the top-level of your Spartacus checkout and run *grunt start*. Doing this means you're using the Spartacus dev-server to serve Spartacus' static assets.

Should you need to customise where Spartacus is accessed from, you can modify the webpay setting *SPARTACUS_STATIC*.

Overriding Spartacus settings via Webpay

To be able to change the settings that Spartacus uses in development you can override *SPA_SETTINGS* in webpay. Any settings here are merged with the settings.js in Spartacus.

2.6.2 Configuring devices

Out of the box, Firefox OS only ships with settings that let you make payments against the production server. If you want to pay with a hosted *dev* or *stage* server then you'll need to put some custom settings on your B2G device. See the developer docs if you want to host your own WebPay.

Using the Firefox OS Simulator

This is **the recommended approach** for developers.

As of Firefox OS Simulator 1.3 you can use a cusom Gaia profile which is necessary to work with a local Webpay server. Skip down to "Build A Custom B2G Profile".

Set up the Firefox OS Simulator according to the documentation. This involves installing a version specific addon, such as a 1.4 Simulator. After installation, open `about:addons` in Firefox and enter the Preferences section for the Simulator add-on. Click the button to use a custom Gaia profile and select the directory of the one you just built.

Open the App Manager, connect to your Simulator and you are ready to test payments against your local Webpay server.

Set Up A Device With ezboot

All you need to do to start testing web payments on a device is flash a recent build, install certs for permissions, push custom settings, and install the Marketplace dev/stage apps.

With ezboot you can do all of this with some commands. First, install ezboot so that the command line script is available on your path.

Now, grab the Webpay source to get the settings you need:

```
git clone git://github.com/mozilla/webpay.git
```

Change into the source directory and set up ezboot:

```
cd webpay
cp ezboot.ini-dist ezboot.ini
```

If you want to make things easier, you can edit `ezboot.ini` and uncomment the wifi and flash settings (i.e. delete the hash prefix). You can add your WiFi details to automatically connect to your local network and add a flash username/password (your LDAP credentials) for faster downloads.

Plug in your device. If this is your *first* time flashing an engineering build (with Marionette), make sure Remote Debugging is enabled in Settings > Device Information > More Information > Developer.

Make sure you're still in the webpay directory and flash the latest build:

```
ezboot flash
```

Set up WiFi:

```
ezboot setup
```

Ask someone for a cert file (see this issue), download the file, and unzip it. Push the dev certs to your device:

```
ezboot mkt_certs --dev --certs_path ~/Downloads/certdb.tmp/
```

Install the packaged Marketplace app:

```
ezboot install_mkt --dev
```

At this time, you need to use the hosted version of Marketplace Stage (not packaged). Install it using the manifest, like this:

```
ezboot install --manifest https://marketplace.allizom.org/manifest.webapp
```

Launch either Marketplace Dev or Marketplace Stage, search for a paid app such as Private Yacht, and click purchase.

That's it! You can stop reading this document because everything else is intended for using custom builds and/or custom settings.

Build A Custom B2G Profile

You have to build a custom profile from the Gaia source to allow `navigator.mozPay()` to talk to your local WebPay server. Refer to the Developing Gaia page for more details but this page has everything you need to know.

IMPORTANT: You have to use a branch of Gaia that matches the version of B2G you're using. For example, check out `origin/v1.2` for 1.2, `origin/v1.4` for 1.4, etc.

Here's an example of building a 1.4 profile. Install git and type these commands:

```
git clone git://github.com/mozilla-b2g/gaia.git
cd gaia
git checkout --track -b origin/v1.4 origin/v1.4
```

Get updates like this:

```
git checkout origin/v1.4
git pull
```

Create `build/config/custom-prefs.js` in that directory. With a text editor, add **all** of the settings below.

IMPORTANT: Before 1.4, you had to put the file in `build/custom-prefs.js`.

Add some basic debug settings:

```
pref("dom.payment.skipHTTPSCheck", true);
pref("dom.identity.enabled", true);
pref("toolkit.identity.debug", true);
```

Add this to activate the hosted dev server:

```
pref("dom.payment.provider.1.name", "firefoxmarketdev");
pref("dom.payment.provider.1.description", "marketplace-dev.allizom.org");
pref("dom.payment.provider.1.uri", "https://marketplace-dev.allizom.org/mozpay/?req=");
pref("dom.payment.provider.1.type", "mozilla-dev/payments/pay/v1");
pref("dom.payment.provider.1.requestMethod", "GET");
```

Add this to activate the hosted stage server:

```
pref("dom.payment.provider.2.name", "firefoxmarketstage");
pref("dom.payment.provider.2.description", "marketplace.allizom.org");
pref("dom.payment.provider.2.uri", "https://marketplace.allizom.org/mozpay/?req=");
pref("dom.payment.provider.2.type", "mozilla-stage/payments/pay/v1");
pref("dom.payment.provider.2.requestMethod", "GET");
```

Add this to activate a local server (make sure the URL is correct for you):

```
pref("dom.payment.provider.3.name", "firefoxmarketlocal");
pref("dom.payment.provider.3.description", "localhost");
pref("dom.payment.provider.3.uri", "http://localhost:2601/mozpay/?req=");
pref("dom.payment.provider.3.type", "mozilla-local/payments/pay/v1");
pref("dom.payment.provider.3.requestMethod", "GET");
```

Add this to activate the payments-alt server:

```
pref("dom.payment.provider.4.name", "firefoxmarketalt");
pref("dom.payment.provider.4.description", "payments-alt.allizom.org");
pref("dom.payment.provider.4.uri", "https://payments-alt.allizom.org/mozpay/?req=");
pref("dom.payment.provider.4.type", "mozilla-alt/payments/pay/v1");
pref("dom.payment.provider.4.requestMethod", "GET");
```

Save the file. Now when you make a profile it will create a `profile/user.js` file with those extra prefs. Type this in the `gaia` directory:

```
make clean profile
```

You now have a custom B2G profile in your `gaia/profile` directory.

These settings are available in the webpay repository: https://github.com/mozilla/webpay/blob/master/ezboot/custom-prefs.js

Setting Up A B2G Device

After you create a custom B2G profile as described above you'll need to flash B2G on your phone and push some profile settings to it.

First make sure you have the Android Developer Tools installed. The `adb` executable should be available in your path.

If you have an Unagi device, you can log in with your Mozilla LDAP credentials and obtain a build from https://pvtbuilds.mozilla.org/pub/mozilla.org/b2g/nightly/mozilla-b2g18-unagi/latest/ At this time, the builds are not available to the public. You could always build your own though.

When you unzip the b2g-distro directory plug your phone in via USB and run this:

```
./flash.sh
```

That installs B2G and Gaia. Before you can add your custom settings you have to enable remote debugging over USB. Go to Settings > Device Information > More Information > Developer and turn on Remote debugging.

Now fetch the gaia code just like in the B2G profile instructions above (make sure you are on the **v1-train** branch), add the `custom-prefs.js` file, and make a custom profile. Here's how to put the custom payment settings on to your phone.

Type these commands:

```
cd gaia
adb shell "stop b2g"
adb push profile/user.js /data/local/
adb reboot
```

When B2G reboots you should be ready to make payments against the configured dev servers Read on to install a Marketplace dev app.

Installing Marketplace Dev

Visit http://app-loader.appspot.com/c5ec6 on your B2G browser to install the Marketplace Dev app. This installs the manifest at https://marketplace-dev.allizom.org/manifest.webapp .

Launch the Marketplace Dev app. If you see pictures of cvan everywhere then you know you've opened the right one. You can set a search filter to show only paid apps. As an example, search for Private Yacht which is fully set up for payments and even checks receipts.

Installing Marketplace Stage

Visit http://app-loader.appspot.com/a2c98 on your B2G browser to install the Marketplace Dev app. This installs the manifest at https://marketplace.allizom.org/manifest.webapp .

Launch the Marketplace Stage app. Search for a paid app such as Private Yacht and make a purchase.

WARNING: the stage app is currently hooked up to the live Bango payment system.

2.7 Packaged Apps

The Marketplace that ships on Firefox OS is a packaged app which needs to be built from fireplace. (Side note: There is a lightweight version of the Marketplace called yogafire which you may also be wanting to build. It follows these same steps, except you'll need to swap out the right manifest URLs when appropriate).

2.7.1 Create a New Package

These directions assume you have a copy of the Marketplace frontend checked out.

If you want to make a production package (i.e., using the production API), simply run from the root of your Fireplace checkout:

```
make package
```

If you are building a packaged app containing an iframed Marketplace (which is currently running in production as of February 2015) as opposed to a true packaged app containing the actual Marketplace and all its assets, you'll instead want to run:

```
make iframe_package
```

Packages will be output to */package/builds/*. You can also make packages for other servers. These are pre-configured within *config.js*:

```
SERVER=altdev make package
SERVER=dev make package
SERVER=paymentsalt make package
SERVER=stage make package
```

If you are running a local backend server (e.g., `http://mp.dev`), you can configure and build a custom package if you wanted. Open *config.js* and add an entry to *packageConfig*. It might look like:

```
'local': {
    domain: 'http://mp.dev',
    media_url: 'http://mp.dev',
    name: 'Local',
    origin: 'app://packaged.local.firefox.com'
}
```

To test your package, use *Firefox's WebIDE*.

2.7.2 Put the New Package on the Marketplace

These directions assume you have the site permissions (meaning: you own the app in the Marketplace) to do this and you want to put it on production. Adjust the URL below if you want to put it on dev or stage.

1. Edit the app on the Marketplace
2. Upload the newly made package
3. Make sure that *Multiple Network Information* and *Network Information* are checked
4. Save your changes. The file will be in the approval queue.

2.7.3 Approve the New Package on the Marketplace

These directions assume you have the reviewer permissions to do this.

1. Load the app in the Reviewer Tools
2. Click *Push to Public*
3. Enter a short message in *comments* and click **OK**

2.7.4 Preload a New Package in Firefox OS

For steps 4 to 8, we have a helper script to download the package and generate Etags.

These directions assume you have forked gaia.

1. File a bug for Gaia saying a new Marketplace is incoming. They won't land a PR without a bug filed.
2. Check out a new branch
3. Navigate to the Marketplace files:

   ```
   cd apps/marketplace.firefox.com
   ```

4. Run in a separate window and make note of the output of these commands. Note that if you are building yogafire and not fireplace you'll want to use *marketplace-package.webapp* from the same URL below.

   ```
   curl 'https://marketplace.firefox.com/packaged.webapp'

   curl -I 'https://marketplace.firefox.com/packaged.webapp' | grep ETag
   ```

5. The first command above will be a blob of JSON including a *package_path*. Copy that package path (eg. *https://marketplace.firefox.com/downloads/file/251994/marketplace-20140331214114.zip*). The second command will have an ETag header (eg. *a10ca98addc3785e92ead363281c425bd7114b84a4162f50096593b76a7ac2c3*) - copy that for later.
6. Replace the current marketplace with the new package using the *package_path* you copied above. Example:

   ```
   curl 'https://marketplace.firefox.com/downloads/file/251994/marketplace-20140331214114.zip' > ap
   ```

7. Get the package's new ETag:

   ```
   curl -I 'https://marketplace.firefox.com/downloads/file/251994/marketplace-20140331214114.zip' |
   ```

8. Modify **metadata.json** and replace the *etag* and *packageEtag* fields with the two ETags from above. Note that the extra escaped quotes are **not** a typo. Your diff will look something like:

   ```
   + "etag": "\"a10ca98addc3785e92ead363281c425bd7114b84a4162f50096593b76a7ac2c3\"",
   + "packageEtag": "\"bf5f4736daffaf7982c872efc4beb38f440d5d84a6fb3f82c5d434cca6abd4d5\"",
   ```

9. Commit your changes and make a pull request. Put the bug number from step 1 in the title of the bug. For example: *Bug 100000 - New Marketplace; 20140501*
10. Return to the bug you filed in step 1. Click *Add Attachment -> Paste text as attachment* and paste in your pull request URL (eg. *https://github.com/mozilla-b2g/gaia/pull/18845*). Under flags request *review?* from *:julienw* or *:fabrice*
11. Done!

2.7.5 Testing Your Package on Device

Do you have that tingle in your gut that says you should test our the package on an actual phone before making a PR? Or maybe you made some changes to the certs and you want to make sure they work? Well, you're in luck, because it's super easy (this will erase everything on your phone):

1. Make sure you're in the root of your Gaia repository
2. Make sure you can see your Flame device with *adb devices*
3. Run *make reset-gaia* to build your gaia and push to the device

2.8 Working With Devices

2.8.1 Flashing the Latest Build

Have ADB installed. You can learn how to install adb on MDN.

Connect your phone to your machine with a USB cable. If you run `adb devices` you should see a device ID.

Flashing the Flame

There are base builds (codename gonk), which contains the kernel, and there are FirefoxOS builds which are run on top of the base build.

Flames with a base build on an old Android kernel (v123) are no longer supported by us. Make sure to run with the newest build.

Note: You will lose everything on the device when you do this. Once you are running v188, you don't need to do this step again until we update to a new kernel (probably a year or two)

Then to run with the correct FirefoxOS build for the flame (requires LDAP):

```
git clone https://github.com/Mozilla-TWQA/B2G-flash-tool
cd B2G-flash-tool
./flash_pvt.py --help
# Shallow flash an engineering build off the master branch onto the
# Flame v188 device.  In theory you won't lose anything...
./flash_pvt.py -d flame-kk -v mozilla-central --eng -g -G --keep_profile
```

Flashing without LDAP Access

- Find the build you want from http://ftp.mozilla.org/pub/mozilla.org/b2g/nightly/ and install (preferably an eng build)
- unzip the file
- navigate to that folder
- run `./flash.sh`
- Phone should be installed with the latest gaia build you chose.

2.8.2 Enabling Dev and Stage Certificates

FxOS version: 1.1 - 1.4

Follow: https://wiki.mozilla.org/Marketplace/Reviewers/Apps/Guide/Setup/Cert_installation

FxOS version: 2.0

You don't need to install certificates; the system preference `dom.mozApps.use_reviewer_certs` just needs to be set to true.

- Connect the device with a USB cable and install any drivers
- Open a shell
- Run `adb shell` and then the following to set the preference:

```
stop b2g
cd /data/b2g/mozilla/\*.default/
echo 'user_pref("dom.mozApps.use_reviewer_certs", true);' >> prefs.js
start b2g
```

FZOS version: 2.1+

Enable the checkbox: *Settings -> Developer -> Use Marketplace reviewer certs*

2.8.3 Installing Dev and Stage on Engineering Builds

Note: Some testers have had trouble with this, and flashing user gaia+gecko fails to boot. Proceed with caution.

Engineering builds (as of 16 Oct 2014) have old Dev and Stage Marketplace apps installed, and they cannot be removed.

To work around this, we need root access on the engineering image in order to push custom prefs (e.g., for payments), and the UI and apps from the user build.

To do so, flash a full engineering image (referred to as "images" as opposed to "gaia+gecko"). Then flash a user build of just "gaia+gecko" of the same build. Since this the user build, the developer menu is hidden by default. To remedy this:

- Go to "Device Information -> More information"
- Scroll to bottom and enable developer menu
- Enable USB Debugging in the dev menu
- Check console enabled
- Enable Marketplace reviewer certs
- Reboot

2.8.4 Accessing Your Local Marketplace From Device

If you want to access your local *Marketplace Backend* on a device, you'll need to proxy the internal virtual server through a public IP and bind that IP to a host on device.

If you run Apache on port 80 which is the default on many systems, you can add this to your config to proxy. Adjust the internal IP address of the virtual server as necessary.:

```
Listen 80

<VirtualHost *:80>
    ServerName mp.dev
    ProxyPreserveHost On
    ProxyRequests off
    ProxyPass / http://192.168.59.103/
    ProxyPassReverse / http://192.168.59.103/
</VirtualHost>
```

If you run nginx on port 80 then you can use a config like this. Again, you may need to adjust the proxied IP:

```
http {
    server {
        listen       80 default;
        server_name  mp.dev;

        location / {
            # Pass public connections to the internal
            # Docker / VirtualBox server.
            proxy_pass http://192.168.59.103/;
            proxy_set_header Host $host;
        }
    }
}
```

When running Docker and serving on your public / network IP (sucha s 10.0.0.1), ensure USB debugging is enabled on your device, plug it in, and use the bind command:

```
bin/mkt bind
```

This will edit the `/system/etc/hosts` file on the device so that you can access http://mp.dev.

If you have multiple network devices, the command will prompt you for the one to bind to. Run `bin/mkt bind --help` for details.

2.8.5 Prefs File for Payments Testing

Here's an example prefs file for payments testing: https://gist.github.com/muffinresearch/9a7c3d3d632a9a9922f0

Push this to your device with:

```
adb push path/to/custom-prefs.js /data/local/user.js
```

Then reboot for the changes to take effect:

```
adb reboot
```

2.8.6 Installing Packaged Marketplaces

Apps such as Dev or Stage or PaymentsAlt, are unlisted on the production Marketplace. Though, Metaplace can be installed which allows you to install the apps from the "jump" menu.

Or you can go directly to the app page from the browser on the device:

- Dev: https://marketplace.firefox.com/app/mkt-dev
- Stage: https://marketplace.firefox.com/app/mkt-stage
- PaymentsAlt: https://marketplace.firefox.com/app/marketplace-payments-alt

2.8.7 Device Not Being Recognized

If your Tarako device is not recognized, add the vendor ID to the ADB USB configuration:

```
echo "0x1782" > ~/.android/adb_usb.ini
adb kill-server
adb start-server
```

These docs are also available as a PDF.