uPaaS Documentation

Release 0.3.0

Łukasz Mierzwa

May 21, 2014

You can report bugs and feature requests on the issues page.

CHAPTER 1

Release notes

After installing current version run the command below to migrate database:

```
upaas_admin migrate_db
```

CHAPTER 2

Demo site

- URL: https://beta.u-paas.org
- Login: guest
- Password: dem0

Contents:

2.1 About uPaaS

uPaaS stands for "uWSGI Platform As A Service", think of it as a "poor man's PaaS". uWSGI itself provides tons of features and uPaaS jobs is to add layer of management on top of it, creating simple, organized and secure way of managing web application deployments.

2.1.1 Features

Currently uPaaS is still in early stages of development so only core features are implemented, but it is designed to provide:

- resource sharing without conflicts - deploy high number of web apps on single box without creating messy environment
- automated scalability - when application is under high load uPaaS will quickly add new nodes
- built in statistics - pretty graphs for every application out of the box
- monitoring and self-repair - uPaaS will constantly check if everything is running and it will try to fix itself when needed
- simple deployments - just do `git push` and uPaaS will take care of the reset (with git post-receive hook)

2.1.2 Components

Database

MongoDB is used to store all data, it provides easy to use replication and provides HA features out of the box.

Router nodes

Routers are load balancers using uWSGI in FastRouter mode.

Backend nodes

Backends are running uPaaS admin web UI and task worker processes.

Notes

uPaaS is still under heavy develoment, many features are missing or incomplete.

2.2 Changelog

2.2.1 0.3.1

Released: 21.05.2014

- improved instance placement - uPaaS is now aware of each backend resources and can pick the best backends for each application instance
- improved backend and router configuration
- improved self healing - uPaaS will now try to detect and fix more instance issues
- Font Awesome updated to 4.1.0

2.2.2 0.3.0

Released: 04.05.2014

- task queue replaced with new task framework
- backbone and tastypie used for ajax
- uWSGI 2.0 packages in PPA
- many improvements under the hood

2.2.3 0.2.1

Released: 06.01.2014

- Added test suite
- Fixed many small bugs found by new tests
- Django 1.6.1 is now used
- Refactored custom domains (db migration needed with `upaas_admin migrate_db` command)

2.2.4 0.2

Released: 22.12.2013

First usable release with most basic functionality implemented.

2.2.5 0.1

Released: 27.10.2013

Technical preview release.

2.3 Installing uPaaS

2.3.1 Pre-Requirements

MongoDB database accessible from backend nodes. MongoDB is used both for uPaaS data and package files, so it might grow to several gigabytes or more, depending on the number of registered application. Each package will use at least 200MB (in case of Ubuntu server).

Domain name that will be pointing to router nodes. You don't need to buy one for testing purposes or intranet usage, if you have dnsmasq running as your dns server you might add those lines to dnsmasq configuration file:

```
address=/upaas.domain/<router node ip>
```

2.3.2 Ubuntu server 12.04 LTS installation

uPaaS is packaged, developed and tested on Ubuntu 12.04 LTS release, but it should be working on any recent Linux distribution. PPA containing uPaaS packages requires some packages that are not available in standard 12.04 release, so few other PPAs must be added to fulfil those dependencies.

Router nodes

Add required PPAs:

```
sudo add-apt-repository ppa:upaas/stable
```

Install upaas-router meta package. It contains uWSGI config files needed to run uWSGI FastRouter node.

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install upaas-router
```

Place custom SSL certificate and key in /etc/upaas/ssl (server.crt and server.key). This step is optional, by default self signed certificate is used.

Backend nodes

Add required PPAs:

```
sudo add-apt-repository ppa:brightbox/ruby-ng
sudo add-apt-repository ppa:fkrull/deadsnakes
sudo add-apt-repository ppa:ondrej/php5
sudo add-apt-repository ppa:upaas/stable
```

Install python-upaas-admin package. It contains uPaaS API and web UI server.

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install python-upaas-admin
```

After installing uPaaS admin package edit `/etc/upaas/upaas.yml` config, it must be identical on all backend nodes. Be sure to set proper storage handler since 0.1.0 release defaults to local file system storage.

See See *uPaaS configuration*.

Once configured uPaaS web UI will be avaiable under http://backend-node-ip/. See `/etc/upaas/upaas_admin_local.ini` for exampled configuration needed to connect uPaaS web UI to upaas-router nodes for high availability.

2.3.3 Create database indexes

```
upaas_admin create_indexes
```

2.3.4 Add first user

Once installed and configured we need to create user with administrator rights:

```
upaas_admin create_user --login john --firstname John --lastname Doe --email john@doe.com --admin
```

2.3.5 Add router node(s)

Login as administrator, go to admin area and create router node(s). Backends will auto-register during task worker startup.

2.4 uPaaS configuration

Configuration file uses YAML syntax. All options are stored in /etc/upaas/upaas.yml configuration file. For convenience settings are split into multiple files that are included in main config (upaas.yml). Splitting uses `!include <path>` statement. If path is not absolute it will be interpreted as relative to the parent file (the one with include statement).

2.4.1 MongoDB settings

Basic settings:

```
mongodb:
  host: localhost
  port: 27017
  database: upaas
```

If MongoDB requires authorization for database access (recommended) add `username` and `password` options.

```
mongodb:
  [...]
  username: username
  password: password
```

To provide high availability to MongoDB installation it is recommended to use MongoDB cluster (replica set or sharding). To pass replica set address to uPaaS use uri option instead of host and port pair, example:

```
mongodb:
  [...]
  uri: mongodb://db1,db2/?replicaSet=upaas&readPreference=primaryPreferred
```

Note: Remeber to set read preference when using HA MongoDB setup.

2.4.2 Admin UI settings

secretkey

For every installation secret key must be set, it must be unique (for each cluster) and unpredictable string. Every node in uPaaS cluster must have identical value secret key.

```
admin:
  secretkey: very-very-secret
```

loglevel

Verbosity level for all logged messages, possible values: debug, info, warning, error.

debug

Enable django debug mode, see django docs for details.

domains

List of domains uPaaS web UI can be served for. Any domain will be allowed if this option is not specified. Details can be found in django docs.

smtp

Contains options for sending email notifications. Avaiable settings:

- `host` - hostname of SMTP server used for sending emails, default is localhost
- `port` - remote SMTP port to use, defult is 25
- `tls` - whenever to use TLS or not, default is false
- `username` - username if SMTP authentication is going to be used, default is not set - not authentication

- `password` - password if SMTP authentication is going to be used, default is not set - not authentication
- `sender` - email address used as sender address, default is no-reply@localhost

Full example:

```
admin:
  secretkey: very-very-secret
  loglevel: info
  debug: false
  domains:
    - "admin.upaas.domain"
    - "admin.upaas.com"
    - "*.upaas-admin.io"
  smtp:
    host: localhost
    port: 25
    tls: true
    username: upaas@localhost
    password: smtppass
```

2.4.3 Paths settings

uPaaS stores files in few location, they can be customized with those settings:

```
paths:
  workdir: /var/upaas/workdir
  apps: /var/upaas/apps
  vassals: /etc/uwsgi-emperor/vassals
```

workdir

Directory for temporary files.

apps

Directory where packages for running applications are stored.

vassals

Directory where applications uWSGI config files are placed. This directory must be the path that uWSGI emperor will be monitoring.

2.4.4 Storage

Package files are stored by default in MongoDB database but custom storage handlers can be created. To use local storage (only useful with single node installations) use those settings:

```
storage:
  handler: upaas.storage.local.LocalStorage
  settings:
    dir: /var/upaas/storage
```

This way uPaaS will store all packages as plain files in /var/upaas/storage directory.

To use dedicated MongoDB database for packages use:

```
storage:
  handler: upaas.storage.mongodb.MongoDBStorage
  settings:
    host: mongo-db-packages-host
    port: 27017
    database: upaas-packages
    username: username
    password: password
```

2.4.5 OS bootstrap

All application packages are built using empty os system image, so first such empty image must be generated. Example config for Ubuntu server:

```
bootstrap:
  timelimit: 600
  env:
    LC_ALL: C
    LANG: C
  commands:
    - debootstrap --components=main,universe,multiverse,restricted `lsb_release -sc` %workdir%
  maxage: 7
  packages:
    - python-software-properties
    - build-essential
```

timelimit

How long single command can take before it is killed (in seconds).

env

List of environment variables passed to each command (optional).

commands

List of commands used to create system image files. `%workdir%` macro will be expanded into directory path where image is being created.

maxage

Images older than this value (in days) will be ignored and new image will be generated. This is intended to keep system images current, with all updates applied.

packages

List of packages to install in system image once it is generated.

2.4.6 System commands

This settings are used to tell uPaaS what commands should be used to interact with system images. Mostly how to (un)install packages using system package manager.

```
commands:
  timelimit: 600
  install:
    env:
      DEBIAN_FRONTEND: noninteractive
      LC_ALL: C
      LANG: C
    cmd: apt-get install --no-install-recommends -y %package%
  uninstall:
    env:
      DEBIAN_FRONTEND: noninteractive
      LC_ALL: C
      LANG: C
    cmd: apt-get remove -y %package%
```

install

Describes how to install package. `cmd` option contains command that needs to be executed, `%package%` macro will be expanded into package name. `env` and `timelimit` options have the same meaning as in bootstrap section.

uninstall

Same as `install` but describes how to uninstall package.

2.4.7 Application deployment settings

uid

Uid of user application will be running as, for example www-data.

gid

Name of group that will be used to run application.

home

Path where application directory will be placed inside system image.

domains

Every application will be accessible using:

- automatic system domain (use `app.domains.system` key to set it)
- custom domain assigned by the user (user must own this domain or at least be able to modify it)

All domains used in application must point to uPaaS router nodes, user will be notified during custom domain assigment. To protect from domain hijacking every custom domain that user want to assign to his application must be verified. This is done by checking if domain contains TXT record with application key. This can be disabled if only trusted apps are deployed in uPaaS cluster, set `apps.domains.validation = False` to disable it.

tcp

Contains two options `port_min` and `port_max` used to specify port range used for application sockets.

Example:

```
apps:
  uid: www-data
  gid: www-data
  home: /home/app
  domain: upaas.domain
  tcp:
    port_min: 2001
    port_max: 7999
```

uwsgi

Contains uWSGI specific options. Currently only `safe_options` section is available with list of safe uWSGI options that user can set in application metadata. Safe options should can be written as python compatible regular expressions.

Example:

```
uwsgi:
  safe_options:
    - "^check-static"
    - "^static-"
    - "^harakiri"
    - "^enable-threads$"
    - "^(worker-|)reload-mercy$"
    - "^max-requests$"
    - "^(min|max)-worker-lifetime$"
    - "^upload-progress$"
    - "^lazy"
    - "^route"
    - "^(response|final|error)-route$"
```

graphite

Configuration for carbon/graphite statistics integration. This is optional feature and it requires working carbon server and graphite frontend applications. Available options: # `carbon` list of carbon servers for pushing statistics from uWSGI backends # `render_url` graphite frontend url, it will be used for rendering statistics, must be accessible by uPaaS users # `timeout` timeout for pushing statistics from uWSGI backend, default is 3 seconds # `frequency` push statistics from uWSGI to carbon every N seconds, default is 60 # `max_retry` how many times uWSGI should retry pushing stats in case of errors, default is 1 # `retry_delay` how many seconds should uWSGI wait before retry, default is 7 # `root` root node for all statistics, default is "uwsgi"

Only `carbon` and `render_url` options are required to integrate carbon/graphite with uPaaS.

2.4.8 Defaults

Default settings. Currently only `limits` section is available. Those limits will be used for all users that do not have custom limits set by uPaaS administrator.

`running_apps`

Numer of applications user is allowed to run simultaneously, 0 means no limit. Default is 0.

`workers`

Number of workers user is allowed to allocate to running applications, 0 means no limit. Default is 0.

`memory_per_worker`

Memory limit for application workers, this limit is applied to each worker process. Default is 128.

`packages_per_app`

Number of package files that are kept for every applications, allowing to rollback application to previous package. Default is 5.

`max_log_size`

Maximum application instance log size (in MB). Each instance logs to file in application home directory (upaas.log), once size limit is reached log is rotated, one rotated log is kept.

Example:

```
limits:
  running_apps: 0
  workers: 16
  memory_per_worker: 128
  packages_per_app: 5
  max_log_size: 50
```

2.4.9 Interpreter settings

Every available interpreter must be configured before app can use it. See `interpreters`.

2.5 CLI client

2.5.1 Installation

```
sudo add-apt-repository ppa:upaas/stable
sudo apt-get update
sudo apt-get install python-upaas-client
```

2.5.2 Configuration file

Before we can start using uPaaS client we must create configuration file:

```
server:
  url: http://admin.upaas.domain
  login: john
  apikey: 98f2ec25ffc9d57e7f168d7c1798c51da1c6a7e3
```

url

URL under which we can access our uPaaS admin web UI.

login

Your user login.

apikey

API key generated for your user account.

Example usage

Deploying redmine:

```
upaas register redmine path/to/redmine.yml
upaas build redmine
[wait until package is build]
upaas start redmine --workers-min 1 --workers-max 4
```

We will have redmine instance with 1 to 4 processes (number scaled based on current load).

2.6 Package building internals

This page describe all steps during package building.

2.6.1 Step 1 - System image

Before we can start building package uPaaS needs to generate bare system image. Under Ubuntu server this is done by using debootstrap command. Once system image is created it is unpacked in temporary directory.

This step can be only customized by uPaaS administrator as it requires tweaking uPaaS configuration file. See *uPaaS configuration* for details.

All commands executed later are run chrooted in this system image.

Empty system image is used only when building first package or if user requests to create fresh package, in other cases current application package is used to build next packages. This way we don't need to reinstall everything all the time.

2.6.2 Step 2 - Install interpreter

This step is executed only for first or fresh package.

uPaaS installs all packages required for interpreter used by our application. This step can be customized only be uPaaS administrator.

2.6.3 Step 3 - Install application dependencies

uPaaS installs all packages requested to install in application metadata file. This step can be customized only using application metadata.

2.6.4 Step 4 - Cloning application

If this is first or fresh package uPassS will try to clone application, otherwise it will only try to update it to the latest revision. This step can be customized by uPaaS administrator, application metadata might override it with custom commands.

2.6.5 Step 5 - Creating files

All files present in metadata `files` section are created.

2.6.6 Step 6 - Executing setup actions

Every type of application requires different commands to be deployed properly, for example:

- ruby needs to run `bundle install`
- python needs `pip install -r requirements.txt`

uPaaS handles that by executing few customizable actions:

- before - this action is intended for application, it allows metadata author to inject any commands needed to be executed before main action, but uPaaS admin provide default commands for this action if needed
- main - this action should be used to run all main actions for given interpreter (like bundler or pip commands), application can override it in metadata file if really needed
- after - this action is intended for application, they should call database migration commands here, uPaaS administrator can provide default actions in configuration file
- finalize - this action can only be customized by uPaaS administrator, it is intended to run cleanup commands

2.6.7 Step 7 - Creating package archive

Compressed tar archive will be created and uploaded to configured storage handler. This can be customized only by uPaaS administrator.

2.7 Metadata format

2.7.1 Syntax

Configuration file uses YAML syntax.

Distribution specific settings, currently only list of packages to install can be configured here.
< id > is the id of the distribution same as output of *lsb_release -si* command.

```
os:
  < id >:
    packages:
      - < package name >
```

Configuration for interpreter used in application.
`type` can be any supported type (python, ruby, php).
You can provide list of interpreter versions supported by your application, highest supported version will be used.
`settings` key allows to pass interpreter specific options, uPaaS administator should document what options are avilable here.
By default only module option for python interpreter can be set under settings key, uPaaS administrator might add support for more options.

```
interpreter:
  type: < type >
  versions:
    - < version >
    - < version >
  settings:
    < interpreter settings >
```

Configuration for repository life cycle management:

- clone - how to clone repository, allowed variables: %destination% - path where application should be cloned to
- update - how to fetch updates

```
repository:
  clone: < command to clone application repository >
  update:
    - < command >
    - < command >
```

After cloning or updating repository uPaaS will try to extract some information about current revision, this should work out of the box for:

- git
- bazaar
- mercurial
- subversion

For any other repository type user can set custom commands used for revision information extraction:

```
repository:
  revision:
    id: <command>
    author: <command>
    date: <command>
    description: <command>
    changelog: <command>
```

Command description:

- id - must return current revision id
- author - must return current revision author
- date - must return current revision date, date must be in any format parsable using timestring module
- description - must return current revision description
- changelog - must return list of changes between two revisions, string `%old%` will be substituted with previous revision id, string `%new%` will be substituted with current revision id

List of environment variables that should be set for this application, optional.

```
env:
  MYENV: value
```

After cloning repository multiple actions needs to be executed in order to deploy application, see *Package building internals* for details.

```
actions:
  setup:
    main:
      - < command >
      - < command >
```

List of files to create after cloning app repository.

```
files:
  < path>: < content >
```

Under `uwsgi` key you can pass additional settings to uWSGI.
uPaaS administrator might limit options that can be set here, unsupported options will be ignored.

```
uwsgi:
  settings:
    - "option = value"
```

Cron tasks. By default cron command will be executed on every running instance, to run it only on one instance set `singleton` to `true`. (Work in progress, not available yet)

```
cron:
  - command: <command>
    minute: <0-59>
    hour: <0-24>
```

```
    day: <1-31>
    month: <1-12>
    weekday: <0-7>
    singelton: <true|false>
  - command <command 2>
    [...]
```

2.7.2 Example

See *Example metadata*

2.8 Example metadata

2.8.1 Note

Current version of metadata examples can be found at github.

2.8.2 Ruby On Rails app

In this example redmine will be deployed.

```
# packages needed on Ubuntu and Debian
os:
Debian: &debian
  packages:
    - git-core
    - rake
    - libpq-dev
    - libmysqlclient-dev
    - libsqlite3-dev
    - imagemagick
    - libmagickcore-dev
    - libmagickwand-dev
Ubuntu: *debian

# current redmine will run with any recent ruby verison
# all supported versions can be found on redmine wiki
# uPaaS will use highest version supported by both redmine and uPaaS itself
interpreter:
type: ruby
versions:
  - 2.0.0
  - 1.9.3

# fetch redmine code from github repo
repository:
clone: git clone --depth=10 --quiet --branch 2.4-stable git://github.com/redmine/redmine.git %destina
update:
  - rm -f Gemfile.lock
  - git reset --hard
  - git pull
info: git log -n 1
changelog: git log --no-merges %old%..%new%"
```

```
# REDMINE_LANG sets default language that redmine will use for UI and for seeding db with sample data
env:
REDMINE_LANG: en

# after installing all gems run after.sh script
actions:
setup:
  after: /bin/bash /tmp/after.sh

# create two files
# after.sh - script that will execute all commands required to deploy redmine
# config/database.yml - redmine database config file, modify it as needed
files:
  /tmp/after.sh: |
    if [ -n "$UPAAS_FRESH_PACKAGE" ] ; then
      echo ">>> Generating secret token"
      rake generate_secret_token || exit 1
    fi
    echo ">>> Migrating database"
    rake db:migrate || exit 1
    if [ -n "$UPAAS_FRESH_PACKAGE" ] ; then
      echo ">>> Loading default data"
      rake redmine:load_default_data || exit 1
    fi
  config/database.yml: |
    production:
      adapter: postgresql
      database: redmine
      host: < db host >
      username: redmine
      password: "< password >"
```

2.8.3 Django app

In this example Django jQuery File Upload demo application will be deployed.

```
# only git is needed to clone application repository from github
os:
  Debian: &debian
    packages:
      - git-core
  Ubuntu: *debian

# both 2.7 and 3.x is supported (needs django == 1.5)
# application will be deployed using Django built in WSGI handler module
interpreter:
  type: python
  versions:
    - "3.2"
    - "2.7"
  settings:
    module: django.core.handlers.wsgi:WSGIHandler()

# clone repository from github
repository:
  clone: git clone --depth=10 --quiet git://github.com/sigurdga/django-jquery-file-upload.git %destin
```

```
  update:
    - git reset --hard
    - git pull
  info: git log -n 1
  changelog: git log --no-merges %old%..%new%"

# Django needs to be told how to load settings module
env:
  DJANGO_SETTINGS_MODULE: django-jquery-file-upload.settings

# django 1.5 is required, 1.6 is not yet supported
# we also symlink app directory since /home will be added to python modules path
# so with this symlink python can load our app as django-jquery-file-upload module
actions:
  setup:
    main:
      - ln -sf /home/app /home/django-jquery-file-upload
      - pip install "django<1.6"
      - pip install pillow
      - python manage.py syncdb --noinput

# pass additional settings to uWSGI so that all it will handle all requests for static files
uwsgi:
  settings:
    - "static-map = /static=fileupload/static"
```

2.8.4 PHP app

In this example phpmyadmin will be deployed.

```
os:
  Debian: &debian
    packages:
      - git-core
      - php5-mysql
      - php5-mcrypt
  Ubuntu: *debian

interpreter:
  type: php
  versions:
    - "5.5"

repository:
  clone: git clone --depth=10 --quiet --branch STABLE git://github.com/phpmyadmin/phpmyadmin.git %des
  update:
    - git reset --hard
    - git pull
  info: git log -n 1
  changelog: git log --no-merges %old%..%new%"

files:
  config/config.inc.php: |
    <?php
    $cfg['blowfish_secret'] = 'changeme';
    $i = 0;
    $i++;
```

```
$cfg['Servers'][$i]['auth_type'] = 'cookie';
$cfg['Servers'][$i]['host'] = 'localhost';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['compress'] = false;
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = false;
$cfg['UploadDir'] = '';
$cfg['SaveDir'] = '';
?>
```

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*