tsuru Documentation

Release 0.5.0

timeredbull

July 01, 2014

Contents

CHAPTER 1

Using tsuru

Deploy your applications on tsuru.

- *command-line installation guide*
- *quickstart*

Building your application:

- *python/django*
- *ruby/rails*
- *php*
- *go*

Other topics:

- *using buildpacks*
- *Procfile syntax*
- *understanding deployment hooks*
- *understanding unit states*
- *recovery and troubleshooting*

1.1 tsuru command-line

- *usage guide*
- *using and creating command-line plugins*

1.2 services

- *using services*
- *install mysql service*

CHAPTER 2

Running tsuru

Build your cloud with tsuru.

- *tsuru architecture*
- *docker provisioner architecture*

2.1 Install tsuru

- *with docker on ubuntu*
- *docker schedulers*

2.2 Configure and run tsuru

- *configuration reference*
- *creating tsuru platforms*
- *backing up tsuru*

2.3 tsuru-admin command-line

- *usage guide*

CHAPTER 3

Extending tsuru

Create your services and use tsuru api.

- *API reference*
- *building your service*
- *crane usage guide*
- *services api workflow*

CHAPTER 4

Get Involved

- *community*

CHAPTER 5

Contribute to tsuru

- *how to contribute*
- *coding style*
- *Building a fully functional development environment with Vagrant*
- *setting up your local environment*

5.1 Creating a platform to tsuru

5.1.1 Overview

Tsuru allows you create apps the way you want using the platform you want. But for that you need to be create a platform prepared for run with.

To Tsuru be able to use your platform you need to configure the following scripts on **/var/lib/tsuru**:

- deploy
- restart
- start

Tsuru has a base platform that you can use to base yours.

5.1.2 Using docker

Now we will create a whole new platform with docker, circus and tsuru basebuilder. Tsuru basebuilder provides to us some useful scripts like **install, setup and start**.

So, using the base platform provided by tsuru we can write a Dockerfile like that:

```
from        ubuntu:14.04
run apt-get install wget -y --force-yes
run wget http://github.com/tsuru/basebuilder/tarball/master -O basebuilder.tar.gz --no-check-certific
run mkdir /var/lib/tsuru
run tar -xvf basebuilder.tar.gz -C /var/lib/tsuru --strip 1
run cp /var/lib/tsuru/base/restart /var/lib/tsuru
run cp /var/lib/tsuru/base/start /var/lib/tsuru
run cp /home/your-user/deploy /var/lib/tsuru
run /var/lib/tsuru/base/install
run /var/lib/tsuru/base/setup
```

5.1.3 Adding your platform to tsuru

If you create a platform using docker, you can use the tsuru-admin cmd to add that.

```
$ tsuru-admin platform-add your-platform-name --dockerfile http://url-to-dockerfile
```

5.2 API reference

5.2.1 1. Endpoints

1.1 Apps

List apps

- Method: GET
- URI: /apps
- Format: json

Returns 200 in case of success, and json in the body of the response containing the app list.

Example:

```
GET /apps HTTP/1.1
Content-Length: 82
[{"Ip":"10.10.10.10","Name":"app1","Units":[{"Name":"app1/0","State":"started"}]}]
```

Info about an app

- Method: GET
- URI: /apps/:appname
- Format: json

Returns 200 in case of success, and a json in the body of the response containing the app content.

Example:

```
GET /apps/myapp HTTP/1.1
Content-Length: 284
{"Name":"app1","Framework":"php","Repository":"git@git.com:php.git","State":"dead", "Units":[{"Ip":"1
```

Remove an app

- Method: DELETE
- URI: /apps/:appname

Returns 200 in case of success.

Example:

```
DELETE /apps/myapp HTTP/1.1
```

Create an app

- Method: POST
- URI: /apps
- Format: json

Returns 200 in case of success, and json in the body of the response containing the status and the url for git repository.

Example:

```
POST /apps HTTP/1.1
{"status":"success", "repository_url":"git@tsuru.plataformas.glb.com:ble.git"}
```

Restart an app

- Method: GET
- URI: /apps/<appname>/restart

Returns 200 in case of success.

Example:

```
GET /apps/myapp/restart HTTP/1.1
```

Get app environment variables

- Method: GET
- URI: /apps/<appname>/env

Returns 200 in case of success, and json in the body returning a dictionary with environment names and values..

Example:

```
GET /apps/myapp/env HTTP/1.1
[{"name": "DATABASE_HOST", "value": "localhost", "public": true}]
```

Set an app environment

- Method: POST
- URI: /apps/<appname>/env

Returns 200 in case of success.

Example:

```
POST /apps/myapp/env HTTP/1.1
```

Delete an app environment

- Method: DELETE
- URI: /apps/<appname>/env

Returns 200 in case of success.

Example:

```
DELETE /apps/myapp/env HTTP/1.1
```

Swapping two apps

- Method: PUT
- URI: /swap?app1=appname&app2=anotherapp

Returns 200 in case of success.

Example:

```
PUT /swap?app1=myapp&app2=anotherapp
```

1.2 Services

List services

- Method: GET
- URI: /services
- Format: json

Returns 200 in case of success.

Example:

```
GET /services HTTP/1.1
Content-Length: 67
{"service": "mongodb", "instances": ["my_nosql", "other-instance"]}
```

Create a new service

- Method: POST
- URI: /services
- Format: yaml
- Body: a yaml with the service metadata.

Returns 200 in case of success. Returns 403 if the user is not a member of a team. Returns 500 if the yaml is invalid. Returns 500 if the service name already exists.

Example:

```
POST /services HTTP/1.1
Body:
    `id: some_service
endpoint:
   production: someservice.com`
```

Remove a service

- Method: DELETE
- URI: /services/<servicename>

Returns 204 in case of success. Returns 403 if user has not access to the server. Returns 403 if service has instances. Returns 404 if service is not found.

Example:

```
DELETE /services/mongodb HTTP/1.1
```

Update a service

- Method: PUT
- URI: /services
- Format: yaml
- Body: a yaml with the service metadata.

Returns 200 in case of success. Returns 403 if the user is not a member of a team. Returns 500 if the yaml is invalid. Returns 500 if the service name already exists.

Example:

```
PUT /services HTTP/1.1
Body:
    `id: some_service
endpoint:
    production: someservice.com`
```

Get info about a service

- Method: GET
- URI: /services/<servicename>
- Format: json

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
GET /services/mongodb HTTP/1.1
[{"Name": "my-mongo", "Teams": ["myteam"], "Apps": ["myapp"], "ServiceName": "mongodb"}]
```

Get service documentation

- Method: GET
- URI: /services/<servicename>/doc
- Format: text

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
GET /services/mongodb/doc HTTP/1.1
Mongodb exports the ...
```

Update service documentation

- Method: PUT
- URI: /services/<servicename>/doc
- Format: text
- Body: text with the documentation

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
PUT /services/mongodb/doc HTTP/1.1
Body: Mongodb exports the ...
```

Grant access to a service

- Method: PUT
- URI: /services/<servicename>/<teamname>

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
PUT /services/mongodb/cobrateam HTTP/1.1
```

Revoke access from a service

- Method: DELETE
- URI: /services/<servicename>/<teamname>

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
DELETE /services/mongodb/cobrateam HTTP/1.1
```

1.3 Service instances

Add a new service instance

- Method: POST
- URI: /services/instances
- Body: *{“name”: “mymysql”: “service_name”: “mysql”}*

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
POST /services/instances HTTP/1.1
{"name": "mymysql": "service_name": "mysql"}
```

Remove a service instance

- Method: DELETE
- URI: /services/instances/<serviceinstancename>

Returns 200 in case of success. Returns 404 if the service does not exists.

Example:

```
DELETE /services/instances/mymysql HTTP/1.1
```

Bind a service instance with an app

- Method: PUT
- URI: /services/instances/<serviceinstancename>/<appname>
- Format: json

Returns 200 in case of success, and json with the environment variables to be exported in the app environ. Returns 403 if the user has not access to the app. Returns 404 if the application does not exists. Returns 404 if the service instance does not exists.

Example:

```
PUT /services/instances/mymysql/myapp HTTP/1.1
Content-Length: 29
{"DATABASE_HOST":"localhost"}
```

Unbind a service instance with an app

- Method: DELETE
- URI: /services/instances/<serviceinstancename>/<appname>

Returns 200 in case of success. Returns 403 if the user has not access to the app. Returns 404 if the application does not exists. Returns 404 if the service instance does not exists.

Example:

```
DELETE /services/instances/mymysql/myapp HTTP/1.1
```

List all services and your instances

- Method: GET
- URI: /services/instances
- Format: json

Returns 200 in case of success and a json with the service list.

Example:

```
GET /services/instances HTTP/1.1
Content-Length: 52
[{"service": "redis", "instances": ["redis-globo"]}]
```

Get an info about a service instance

- Method: GET
- URI: /services/instances/<serviceinstancename>
- Format: json

Returns 200 in case of success and a json with the service instance data. Returns 404 if the service instance does not exists.

Example:

```
GET /services/instances/mymysql HTTP/1.1
Content-Length: 71
{"name": "mongo-1", "servicename": "mongodb", "teams": [], "apps": []}
```

service instance status

- Method: GET
- URI: /services/instances/<serviceinstancename>/status

Returns 200 in case of success.

Example:

```
GET /services/instances/mymysql/status HTTP/1.1
```

1.4 Quotas

Get quota info of an user

- Method: GET
- URI: /quota/<user>
- Format: json

Returns 200 in case of success, and json with the quota info.

Example:

```
GET /quota/wolverine HTTP/1.1
Content-Length: 29
{"items": 10, "available": 2}
```

1.5 Healers

List healers

- Method: GET
- URI: /healers
- Format: json

Returns 200 in case of success, and json in the body with a list of healers.

Example:

```
GET /healers HTTP/1.1
Content-Length: 35
[{"app-heal": "http://healer.com"}]
```

Execute healer

- Method: GET
- URI: /healers/<healer>

Returns 200 in case of success.

Example:

```
GET /healers/app-heal HTTP/1.1
```

1.6 Platforms

List platforms

- Method: GET
- URI: /platforms
- Format: json

Returns 200 in case of success, and json in the body with a list of platforms.

Example:

```
GET /platforms HTTP/1.1
Content-Length: 67
[{Name: "python"},{Name: "java"},{Name: "ruby20"},{Name: "static"}]
```

1.7 Users

Create an user

- Method: POST
- URI: /users
- Body: *{"email":"nobody@globo.com","password":"123456"}*

Returns 200 in case of success. Returns 400 if the json is invalid. Returns 400 if the email is invalid. Returns 400 if the password characters length is less than 6 and greater than 50. Returns 409 if the email already exists.

Example:

```
POST /users HTTP/1.1
Body: '{"email":"nobody@globo.com","password":"123456"}'
```

Reset password

- Method: POST
- URI: /users/<email>/password?token=token

Returns 200 in case of success. Returns 404 if the user is not found.

The token parameter is optional.

Example:

```
POST /users/user@email.com/password?token=1234 HTTP/1.1
```

Login

- Method: POST
- URI: /users/<email>/tokens
- Body: *{"password":"123456"}*

Returns 200 in case of success. Returns 400 if the json is invalid. Returns 400 if the password is empty or nil. Returns 404 if the user is not found.

Example:

```
POST /users/user@email.com/tokens HTTP/1.1
{"token":"e275317394fb099f62b3993fd09e5f23b258d55f"}
```

Logout

- Method: DELETE
- URI: /users/tokens

Returns 200 in case of success.

Example:

```
DELETE /users/tokens HTTP/1.1
```

Change password

- Method: PUT
- URI: /users/password
- Body: *{"old":"123456","new":"654321"}*

Returns 200 in case of success. Returns 400 if the json is invalid. Returns 400 if the old or new password is empty or nil. Returns 400 if the new password characters length is less than 6 and greater than 50. Returns 403 if the old password does not match with the current password.

Example:

```
PUT /users/password HTTP/1.1
Body: `{"old":"123456","new":"654321"}`
```

Remove an user

- Method: DELETE
- URI: /users

Returns 200 in case of success.

Example:

```
DELETE /users HTTP/1.1
```

Add public key to user

- Method: POST
- URI: /users/keys
- Body: *{"key":"my-key"}*

Returns 200 in case of success.

Example:

```
POST /users/keys HTTP/1.1
Body: `{"key":"my-key"}`
```

Remove public key from user

- Method: DELETE
- URI: /users/keys
- Body: *{"key":"my-key"}*

Returns 200 in case of success.

Example:

```
DELETE /users/keys HTTP/1.1
Body: `{"key":"my-key"}`
```

1.8 Teams

List teams

- Method: GET

- URI: /teams
- Format: json

Returns 200 in case of success, and json in the body with a list of teams.

Example:

```
GET /teams HTTP/1.1
Content-Length: 22
[{"name": "teamname"}]
```

Info about a team

- Method: GET
- URI: /teams/<teamname>
- Format: json

Returns 200 in case of success, and json in the body with the info about a team.

Example:

```
GET /teams/teamname HTTP/1.1
{"name": "teamname", "users": ["user@email.com"]}
```

Add a team

- Method: POST
- URI: /teams

Returns 200 in case of success.

Example:

```
POST /teams HTTP/1.1
{"name": "teamname"}
```

Remove a team

- Method: DELETE
- URI: /teams/<teamname>

Returns 200 in case of success.

Example:

```
DELELE /teams/myteam HTTP/1.1
```

Add user to team

- Method: PUT
- URI: /teams/<teanmaname>/<username>

Returns 200 in case of success.

Example:

```
PUT /teams/myteam/myuser HTTP/1.1
```

Remove user from team

- Method: DELETE
- URI: /teams/<teanmaname>/<username>

Returns 200 in case of success.

Example:

```
DELETE /teams/myteam/myuser HTTP/1.1
```

1.9 Tokens

Generate app token

- Method: POST
- URI: /tokens
- Format: json

Returns 200 in case of success, with the token in the body.

Example:

```
POST /tokens HTTP/1.1
    {
            "Token": "sometoken",
            "Creation": "2001/01/01",
            "Expires": 1000,
            "AppName": "appname",
    }
```

5.3 tsuru architecture

5.3.1 api

The api is the heart of *tsuru*. The api is responsible to the deploy workflow and the lifecycle of *apps*.

provisioners

The provisioner is responsible for provision the *units*.

There is only one supported provisioner right now:

- docker

routers

The router routes incoming traffic to the application units.

Currently, there is two routers:

- elb
- hipache

5.3.2 mongodb

tsuru uses *mongodb* to store all data about *apps*, *units*, *services*, *users* and teams.

5.3.3 gandalf

gandalf is a REST api to manage git repositories, users and provide access to them over SSH.

5.4 community

5.4.1 irc channel

#tsuru channel on irc.freenode.net - chat with other tsuru users and developers

5.4.2 ticket system

ticket system - report bugs and make feature requests

5.5 Configuring tsuru

tsuru uses a configuration file in YAML format. This document describes what each option means, and how it should look like.

5.5.1 Notation

tsuru uses a colon to represent nesting in YAML. So, whenever this document say something like `key1:key2`, it refers to the value of the `key2` that is nested in the block that is the value of `key1`. For example, `database:url` means:

```
database:
  url: <value>
```

5.5.2 tsuru configuration

This section describes tsuru's core configuration. Other sections will include configuration of optional components, and finally, a full sample file.

HTTP server

tsuru provides a REST API, that supports HTTP and HTTP/TLS (a.k.a. HTTPS). Here are the options that affect how tsuru's API behaves:

listen

`listen` defines in which address tsuru webserver will listen. It has the form <host>:<port>. You may omit the host (example: `:8080`). This setting has no default value.

use-tls

`use-tls` indicates whether tsuru should use TLS or not. This setting is optional, and defaults to "false".

tls:cert-file

`tls:cert-file` is the path to the X.509 certificate file configured to serve the domain. This setting is optional, unless `use-tls` is true.

tls:key-file

`tls:key-file` is the path to private key file configured to serve the domain. This setting is optional, unless `use-tls` is true.

Database access

tsuru uses MongoDB as database manager, to store information about users, VM's, and its components. Regarding database control, you're able to define to which database server tsuru will connect (providing a MongoDB connection string). The database related options are listed below:

database:url

`database:url` is the database connection string. It is a mandatory setting and has no default value. Examples of strings include the basic "127.0.0.1" and the more advanced "mongodb://user@password:127.0.0.1:27017/database". Please refer to MongoDB documentation for more details and examples of connection strings.

database:name

`database:name` is the name of the database that tsuru uses. It is a mandatory setting and has no default value. An example of value is "tsuru".

Email configuration

tsuru sends email to users when they request password recovery. In order to send those emails, tsuru needs to be configured with some SMTP settings. Omitting these settings won't break tsuru, but users would not be able to reset their password automatically.

smtp:server

The SMTP server to connect to. It must be in the form <host>:<port>. Example: "smtp.gmail.com:587".

smtp:user

The user to authenticate with the SMTP sever. Currently, tsuru requires authenticated sessions.

smtp:password

The password for authentication within the SMTP server.

Git configuration

tsuru uses Gandalf to manage git repositories. Gandalf exposes a REST API for repositories management, and tsuru uses it. So tsuru requires information about the Gandalf HTTP server, and also its git-daemon and SSH service.

tsuru also needs to know where the git repository will be cloned and stored in units storage. Here are all options related to git repositories:

git:unit-repo

`git:unit-repo` is the path where tsuru will clone and manage the git repository in all units of an application. This is where the code of the applications will be stored in their units. Example of value: `/home/application/current`.

git:api-server

`git:api-server` is the address of the Gandalf API. It should define the entire address, including protocol and port. Examples of value: `http://localhost:9090` and `https://gandalf.tsuru.io:9595`.

git:rw-host

`git:rw-host` is the host that will be used to build the push URL. For example, when the value is "tsuruhost.com", the push URL will be something like git@tsuruhost.com:<app-name>.git.

git:ro-host

`git:ro-host` is the host that units will use to clone code from users applications. It's used to build the read only URL of the repository. For example, when the value is "tsuruhost.com", the read-only URL will be something like git://tsuruhost.com/<app-name>.git.

Authentication configuration

tsuru has support for `native` and `oauth` authentication schemes.

The default scheme is `native` and it supports the creation of users in Tsuru's internal database. It hashes passwords brcypt and tokens are generated during authentication, and are hashed using SHA512.

The `auth` section also controls whether user registration is on or off. When user registration is off, the user creation URL is not registered in the server.

auth:scheme

The authentication scheme to be used. The default value is `native`, the other supported value is `oauth`.

auth:user-registration

This flag indicates whether user registration is enabled. This setting is optional, and defaults to false.

auth:hash-cost

Required only with `native` chosen as `auth:scheme`.

This number indicates how many CPU time you're willing to give to hashing calculation. It is an absolute number, between 4 and 31, where 4 is faster and less secure, while 31 is very secure and *very* slow.

auth:token-expire-days

Required only with `native` chosen as `auth:scheme`.

Whenever a user logs in, tsuru generates a token for him/her, and the user may store the token. `auth:token-expire-days` setting defines the amount of days that the token will be valid. This setting is optional, and defaults to "7".

auth:max-simultaneous-sessions

tsuru can limit the number of simultaneous sessions per user. This setting is optional, and defaults to "unlimited".

auth:oauth

Every config entry inside `auth:oauth` are used when the `auth:scheme` is set to "oauth". Please check rfc6749 for more details.

auth:oauth:client-id

The client id provided by your OAuth server.

auth:oauth:client-secret

The client secret provided by your OAuth server.

auth:oauth:scope

The scope for your authentication request.

auth:oauth:auth-url

The URL used in the authorization step of the OAuth flow. Tsuru CLI will receive this URL and trigger the opening a browser on this URL with the necessary parameters.

During the authorization step, Tsuru CLI will start a server locally and set the callback to http://localhost:<port>, if `auth:oauth:callback-port` is set Tsuru CLI will use its value as <port>. If `auth:oauth:callback-port` isn't present Tsuru CLI will automatically choose an open port.

The callback URL should be registered on your OAuth server.

If the chosen server requires the callback URL to match the same host and port as the registered one you should register "http://localhost:<chosen port>" and set the `auth:oauth:callback-port` accordingly.

If the chosen server is more lenient and allows a different port to be used you should register simply "http://localhost" and leave `auth:oauth:callback-port` empty.

auth:oauth:token-url

The URL used in the exchange token step of the OAuth flow.

auth:oauth:info-url

The URL used to fetch information about the authenticated user. Tsuru expects a json response containing a field called `email`.

Tsuru will also make call this URL on every request to the API to make sure the token is still valid and hasn't been revoked.

auth:oauth:collection

The database collection used to store valid access tokens. Defaults to "oauth_tokens".

auth:oauth:callback-port

The port used in the callback URL during the authorization step. Check docs for `auth:oauth:auth-url` for more details.

queue configuration

tsuru uses a work queue for asynchronous tasks.

tsuru supports both `redis` and `beanstalkd` as queue backends. However, using `beanstalkd` is *deprecated* as of 0.5.0. The log live streaming feature "tsuru log -f" will not work if using `beanstalkd`.

For compatibility and historical reasons the default queue is beanstalkd. You can customize the used queue, and settings related to the queue (like the address where the server is listening).

Creating a new queue provider is as easy as implementing an interface.

queue

`queue` is the name of the queue implementation that tsuru will use. This setting defaults to `beanstalkd`, but we strongly encourage you to change it to `redis`.

queue-server

`queue-server` is the TCP address where beanstalkd is listening. This setting is optional and defaults to "localhost:11300".

redis-queue:host

`redis-queue:host` is the host of the Redis server to be used for the working queue. This settings is optional and defaults to "localhost".

redis-queue:port

`redis-queue:port` is the port of the Redis server to be used for the working queue. This settings is optional and defaults to 6379.

redis-queue:password

`redis-queue:password` is the password of the Redis server to be used for the working queue. This settings is optional and defaults to "", indicating that the Redis server is not authenticated.

redis-queue:db

`redis-queue:db` is the database number of the Redis server to be used for the working queue. This settings is optional and defaults to 3.

Admin users

tsuru has a very simple way to identify admin users: an admin user is a user that is the member of the admin team, and the admin team is defined in the configuration file, using the `admin-team` setting.

admin-team

`admin-team` is the name of the administration team for the current tsuru installation. All members of the administration team is able to use the `tsuru-admin` command.

Quota management

tsuru can, optionally, manage quotas. Currently, there are two available quotas: apps per user and units per app.

tsuru administrators can control the default quota for new users and new apps in the configuration file, and use `tsuru-admin` command to change quotas for users or apps. Quota management is disabled by default, to enable it, just set the desired quota to a positive integer.

quota:units-per-app

`quota:units-per-app` is the default value for units per-app quota. All new apps will have at most the number of units specified by this setting. This setting is optional, and defaults to "unlimited".

quota:apps-per-user

`quota:apps-per-user` is the default value for apps per-user quota. All new users will have at most the number of apps specified by this setting. This setting is optional, and defaults to "unlimited".

Log level

debug

`false` is the default value, so you won't see any noises on logs, to turn it on set it to true, e.g.: `debug:  true`

Defining the provisioner

tsuru has extensible support for provisioners. A provisioner is a Go type that satisfies the *provision.Provisioner* interface. By default, tsuru will use `DockerProvisioner` (identified by the string "docker"), and now that's the only supported provisioner (Ubuntu Juju was supported in the past but its support has been removed from Tsuru).

provisioner

`provisioner` is the string the name of the provisioner that will be used by tsuru. This setting is optional and defaults to "docker".

Docker provisioner configuration

docker:collection

Database collection name used to store containers information.

docker:repository-namespace

TODO: see *tsuru with docker </docker>*

docker:router

TODO: see *tsuru with docker </docker>*

docker:deploy-cmd

TODO: see *tsuru with docker </docker>*

docker:ssh-agent-port

TODO: see *tsuru with docker </docker>*

docker:segregate

TODO: see *tsuru with docker </docker>*

docker:scheduler:redis-server

TODO: see *tsuru with docker </docker>*

docker:scheduler:redis-prefix

TODO: see *tsuru with docker </docker>*

docker:run-cmd:bin

TODO: see *tsuru with docker </docker>*

docker:run-cmd:port

TODO: see *tsuru with docker </docker>*

docker:ssh:add-key-cmd

TODO: see *tsuru with docker </docker>*

docker:ssh:public-key

TODO: see *tsuru with docker </docker>*

docker:ssh:user

TODO: see *tsuru with docker </docker>*

5.5.3 Sample file

Here is a complete example:

```
listen: "0.0.0.0:8080"
host: http://{{{API_HOST}}}:8080
admin-team: admin

database:
  url: {{{MONGO_HOST}}}:{{{MONGO_PORT}}}
```

```
  name: tsurudb

git:
  unit-repo: /home/application/current
  api-server: http://{{{GANDALF_HOST}}}:8000
  rw-host: {{{GANDALF_HOST}}}
  ro-host: {{{GANDALF_HOST}}}

auth:
  user-registration: true
  scheme: native

provisioner: docker
hipache:
  domain: {{{HOST_NAME}}}
queue: redis
redis-queue:
  host: localhost
  port: 6379
docker:
  collection: docker_containers
  repository-namespace: tsuru
  router: hipache
  deploy-cmd: /var/lib/tsuru/deploy
  ssh-agent-port: 4545
  segregate: true
  scheduler:
    redis-server: 127.0.0.1:6379
    redis-prefix: docker-cluster
  run-cmd:
    bin: /var/lib/tsuru/start
    port: "8888"
  ssh:
    add-key-cmd: /var/lib/tsuru/add-key
    public-key: /var/lib/tsuru/.ssh/id_rsa.pub
    user: ubuntu
```

5.6 contribute

- Source hosted at GitHub
- Report issues on GitHub Issues

Pull requests are very welcome! Make sure your patches are well tested and documented :)

5.6.1 development environment

See this guide to *setting up you tsuru development environment*.

And follow our *coding style guide*.

5.6.2 running the tests

You can use *make* to install all tsuru dependencies and run tests. It will also check if everything is ok with your *GOPATH* setup:

```
$ make
```

5.6.3 writing docs

tsuru documentation is written using Sphinx, which uses RST. Check these tools docs to learn how to write docs for tsuru.

5.6.4 building docs

In order to build the HTML docs, just run on terminal:

```
$ make doc
```

5.7 Install tsuru and Docker

This document describes how to manually install all tsuru compoments in one virtual machine. Installing all components in one machine is not recommended for production ready but is a good start to have a tsuru stack working.

You can install it automatically using tsuru-now (or tsuru-bootstrap, that runs tsuru-now on vagrant).

tsuru components are composed by:

- MongoDB
- Redis
- Hipache
- Docker
- Gandalf
- tsuru API

This document assumes that tsuru is being installed on a Ubuntu Server 14.04 LTS 64-bit machine.

5.7.1 Before install

Before install, let's install curl and python-software-properties, that are used to install extra repositories.

```
sudo apt-get update
sudo apt-get install curl python-software-properties -qqy
```

5.7.2 Adding repositories

Let's start adding the repositories for Docker, tsuru and MongoDB.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 36A1D7869245C8950F966E92D8576A
echo "deb http://get.docker.io/ubuntu docker main" | sudo tee /etc/apt/sources.list.d/docker.list

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" | sudo tee /etc/apt/sou

sudo apt-add-repository ppa:tsuru/ppa -y
```

```
sudo apt-get update
```

5.7.3 Installing MongoDB

tsuru uses MongoDB to store all data about apps, users and teams. Let's install it:

```
sudo apt-get install mongodb-org -qqy
```

5.7.4 Installing Redis

tsuru uses Redis for message queueing and Hipache uses it for storing routing data.

```
sudo apt-get install redis-server -qqy
```

5.7.5 Installing Hipache

Hipache is a distributed HTTP and websocket proxy. tsuru uses Hipache to route the requests to the containers.

In order to install Hipache, just use apt-get:

```
sudo apt-get install node-hipache -qqy
```

Now let's start Hipache

```
sudo start hipache
```

5.7.6 Installing docker

```
sudo apt-get install lxc-docker -qqy
```

tsuru uses the docker HTTP api to manage the containers, to it works it is needed to configure docker to use tcp protocol.

To change it, edit the */etc/default/docker* adding this line:

```
export DOCKER_OPTS="-H 127.0.0.1:4243"
```

Then restart docker:

```
sudo stop docker
sudo start docker
```

5.7.7 Installing gandalf and archive-server

tsuru uses gandalf to manage git repositories.

```
sudo apt-get install gandalf-server -qqy
```

A deploy is executed in the `git push`. In order to get it working, you will need to add a pre-receive hook. Tsuru comes with three pre-receive hooks, all of them need further configuration:

- s3cmd: uses Amazon S3 to store and server archives

- archive-server: uses tsuru's archive-server to store and serve archives
- swift: uses Swift to store and server archives (compatible with Rackspace Cloud Files)

In this tutorial, we will use archive-server, but you can use anything that can store a git archive and serve it via HTTP or FTP. You can install archive-server via apt-get too:

```
sudo apt-get install archive-server -qqy
```

Then you will need to configure Gandalf, install the pre-receive hook, set the proper environment variables and start Gandalf and the archive-server:

```
hook_dir=/home/git/bare-template/hooks
sudo mkdir -p $hook_dir
sudo curl https://raw.githubusercontent.com/tsuru/tsuru/master/misc/git-hooks/pre-receive.archive-ser
sudo chmod +x ${hook_dir}/pre-receive
sudo chown -R git:git /home/git/bare-template
cat | sudo tee -a /home/git/.bash_profile <<EOF
export ARCHIVE_SERVER_READ=http://172.17.42.1:3232 ARCHIVE_SERVER_WRITE=http://127.0.0.1:3131
EOF
```

In the `/etc/gandalf.conf` file, remove the comment from the line "template: /home/git/bare-template", so it looks like that:

```
git:
  bare:
    location: /var/lib/gandalf/repositories
    template: /home/git/bare-template
```

Then start gandalf and archive-server:

```
sudo start gandalf-server
sudo start archive-server
```

5.7.8 Installing tsuru API server

```
sudo apt-get install tsuru-server -qqy

sudo sed -i -e 's/=no/=yes/' /etc/default/tsuru-server
sudo start tsuru-ssh-agent
sudo start tsuru-server-api
```

Now you need to customize the configuration in the `/etc/tsuru/tsuru.conf`.

```
sudo vim /etc/tsuru/tsuru.conf
```

The basic configuration is:

```
listen: "0.0.0.0:8080"
debug: true
host: http://machine-public-ip:8080 # This port must be the same as in the "listen" conf
admin-team: admin
auth:
    user-registration: true
    scheme: native # you can use oauth or native
database:
    url: 127.0.0.1:27017
    name: tsurudb
queue: redis
```

```
redis-queue:
    host: 127.0.0.1
    port: 6379
```

Now we will configure git:

```
git:
    unit-repo: /home/application/current
    api-server: http://127.0.0.1:8000
```

Finally, we will configure docker:

```
provisioner: docker
docker:
    segregate: false
    servers:
        - http://127.0.0.1:4243
    router: hipache
    collection: docker_containers
    repository-namespace: tsuru
    deploy-cmd: /var/lib/tsuru/deploy
    ssh-agent-port: 4545
    scheduler:
        redis-server: 127.0.0.1:6379
        redis-prefix: docker-cluster
    run-cmd:
        bin: /var/lib/tsuru/start
        port: "8888"
    ssh:
        add-key-cmd: /var/lib/tsuru/add-key
        public-key: /var/lib/tsuru/.ssh/id_rsa.pub
        user: ubuntu
hipache:
    domain: tsuru-sample.com # tsuru uses this to mount the app's urls
```

All confs are better explained here.

5.7.9 Generating token for Gandalf authentication

The last step before is to tell the pre-receive script where to find the tsuru server and how to talk to it. We do that by exporting two environment variables in the `~git/.bash_profile` file:

```
cat | sudo tee -a /home/git/.bash_profile <<EOF
export TSURU_HOST=http://127.0.0.1:8080
export TSURU_TOKEN=`tsr token`
EOF
```

5.7.10 Using tsuru client

Congratulations! At this point you should have a working tsuru server running on your machine, follow the *tsuru client usage guide* to start build your apps.

5.7.11 Installing platforms

After creating the first user and the admin team, you can use the *tsuru-admin* to install your preferred platform:

```
tsuru-admin platform-add platform-name --dockerfile dockerfile-url
```

For example, Python:

```
tsuru-admin platform-add python --dockerfile https://raw.githubusercontent.com/tsuru/basebuilder/mast
```

You can see the oficial tsuru dockerfiles here: https://github.com/tsuru/basebuilder.

Here you can see more docs about tsuru-admin.

5.7.12 Adding Services

Here you will find a complete step-by-step example of how to install a mysql service with tsuru: *Install and configure a MySQL service*.

5.7.13 DNS server

You can integrate any DNS server with tsuru. *Here you can find an example of using bind as a DNS forwarder*, integrated with tsuru.

5.8 Build your own PaaS with tsuru and Docker on Centos

This document describes how to create a private PaaS service using tsuru and docker on Centos.

This document assumes that tsuru is being installed on a Centos (6.4+) machine. You can use equivalent packages for beanstalkd, git, MongoDB and other tsuru dependencies. Please make sure you satisfy minimal version requirements.

Just follow this steps:

5.8.1 DNS server

You can integrate any DNS server with tsuru. Here: http://docs.tsuru.io/en/latest/misc/dns-forwarders.html you can find a example of how to install a DNS server integrated with tsuru

5.8.2 Docker

To make docker working on a RHEL/Centos distro, you will need to use the EPEL repository, build a kernel with AUFS support, and install all dependencies as following:

```
# Installing the EPEL respository
$ rpm -iUvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
$ yum update -y
```

Here you can install our own kernel+lxc+docker or compile them To install our RPM package ready to go:

```
# Installing the EPEL respository
# rpm -iUvh http://tsuru.s3.amazonaws.com/centos/docker-0.6.0-1.el6.x86_64.rpm \
http://tsuru.s3.amazonaws.com/centos/lxc-0.8.0-3.el6.x86_64.rpm \
http://tsuru.s3.amazonaws.com/centos/lxc-libs-0.8.0-3.el6.x86_64.rpm \
http://tsuru.s3.amazonaws.com/centos/kernel-ml-aufs-3.10.11-1.el6.x86_64.rpm
```

To Compile, just follow these steps

```
# Download the kernel + dependencies for docker
$ yum install fedora-packager -y
# you will need to perform these steps bellow with a unprivileged user, ex: su - tsuru
$ git clone https://github.com/sciurus/docker-rhel-rpm
$ cd docker-rhel-rpm
# Remove auto restart of docker, as it will be managed by circus
$ sed -i 's|^%{_sysconfdir}/init/docker.conf||; s/.*source1.*//i' docker/docker.spec
```

Now, just follow the steps to build the kernel + lxc + docker from here: https://github.com/sciurus/docker-rhel-rpm/blob/master/README.md

```
# In order to use docker, you will need to allow the ip forward
$ grep ^net.ipv4.ip_forward /etc/sysctl.conf > /dev/null 2>&1 && \
                    sed -i 's/^net.ipv4.ip_forward.*/net.ipv4.ip_forward = 1/' /etc/sysctl.conf  || \
                    echo 'net.ipv4.ip_forward = 1' >> /etc/sysctl.conf
$ sysctl -p
# You also need to disable selinux, adding the parameter "selinux=0" in your new kernel 3.10 (/boot/g
$ grep selinux=0 /boot/grub/menu.lst
# Turn off your default firewall rules for now
$ service iptables stop
$ chkconfig iptables off
```

After build, install and reboot the server with the new kernel(it will take some time), you will need to install the tsuru's dependencies

5.8.3 tsuru's Dependencies

tsuru needs MongoDB stable, distributed by 10genr, Beanstalkd as work queue, git-daemon(necessary for Gandalf) and Redis for hipache pt-ge Install the latest EPEL version, by doing this:

```
$ yum install mongodb-server beanstalkd git-daemon redis python-pip python-devel gcc gcc-c++ -y
$ service mongod start
$ service beanstalkd start
$ service redis start
$ chkconfig mongod on
$ chkconfig beanstalkd on
$ chkconfig redis on
```

5.8.4 tsuru Setup

tsuru uses Gandalf to manage git repositories, and hipache as router To setup tsuru, just follow this steps. Obs: It can be used to upgrade this services as needed

```
$ curl https://raw.github.com/tsuru/tsuru/master/misc/functions-docker-centos.sh -o functions-docker-
$ source functions-docker-centos.sh
# Install tsuru Server(tsr), Gandalf, Hipache and Circus for monitoring
$ install_services
```

Configuring

Before running tsuru, you must configure it. By default, tsuru will look for the configuration file in the `/etc/tsuru/tsuru.conf` path. You can check a sample configuration file and documentation for each tsuru setting in the *"Configuring tsuru"* page.

You can download the sample configuration file from Github

By default, this configuration will use the tsuru image namespace, so if you try to create an application using python platform, tsuru will search for an image named tsuru/python. You can change this default behavior by changing the docker:repository-namespace config field.

To automatically configure tsuru and all other services, just run the function presented in functions-docker-centos.sh file, as following

```
# It will configure tsuru, gandalf, hipache and circus. If you had already done that before, your pre
$ source functions-docker-centos.sh #you already did it above
$ configure_services_for_first_time
# start circus
$ initctl start circusd
```

At that time, circus should be running and started all the tsuru services

Running

Now that you have `tsr` properly installed, and you *configured tsuru* Verify api and docker-ssh-agent

```
$ ps -ef|grep ts[r]
```

Creating Docker Images

Now it's time to install the docker images for your needeeds platform. You can build your own docker image, or you can use ours own images as following

```
# Add an alias for docker to make your life easier (add it to your .bash_profile)
$ alias docker='docker -H 127.0.0.1:4243'
# Build the wanted platform, here we are adding the static platform(webserver)
$ docker build -t tsuru/static https://raw.github.com/tsuru/basebuilder/master/static/Dockerfile
# Now you can see if your image is ready - you should see the tsuru/static as an repository
$ docker images
# If you want all the other platforms, just run the command bellow
$ for image in nodejs php python ruby; do docker build -t tsuru/$image https://raw.github.com/tsuru/b
# To see if everything went well - just take a look in the repository column
$ docker images
# Now try to create your apps!
```

Using tsuru

Congratulations! At this point you should have a working tsuru server running on your machine, follow the *tsuru client usage guide* to start build your apps.

Adding Services

Here you will find a complete step-by-step example of how to install a mysql service with tsuru: http://docs.tsuru.io/en/latest/services/mysql-example.html

This document describes the installation of a single docker node. It can be use to create a docker cluster to be used by tsuru. At the end of this document, you will have a running and configured docker node.

5.9 Setup

This document assumes you already have a tsuru server installed and running, if you don't, follow the :doc: *installation with docker <docker>*. The docker installation on tsuru server in this case is optional, if you install it in the end you'll have two docker nodes.

Now that you have you setup ready, let's see how to install docker:

5.10 docker

```
$ wget -qO- https://get.docker.io/gpg | sudo apt-key add -
$ echo 'deb http://get.docker.io/ubuntu docker main' | sudo tee /etc/apt/sources.list.d/docker.list
$ sudo apt-get update
$ sudo apt-get install lxc-docker
```

Then edit `/etc/init/docker.conf` to start docker on tcp://0.0.0.0:4243:

```
description     "Docker daemon"

start on filesystem and started lxc-net
stop on runlevel [!2345]

respawn

script
    /usr/bin/docker -H tcp://0.0.0.0:4243 -d
end script
```

Now start it:

```
$ sudo start docker
```

5.11 tsuru node agent

Now that you have docker installed and running it's time to install tsuru node agent. This agent is responsible to announce a docker node, unnannounce it, and run the `docker-ssh-agent`.

Add the tsuru/ppa then install it.

```
$ sudo add-apt-repository -y ppa:tsuru/ppa
$ sudo apt-get update
$ sudo apt-get tsuru-node-agent
```

Start `docker-ssh-agent`:

```
$ start tsuru-node-agent docker-ssh-agent
```

Now it is need to announce the node we just created:

```
$ tsuru-node-agent node-add address=<address> ID=<server-id> team=<team-owner> -h <tsuru-api:port>
```

The `address` parameter is the address of the node we just installed. The `ID` parameter is the identifier for this host that tsuru will use. The `team` parameter is the team that this host will attend, see the scheduler docs to know more about it. This parameter is optional, if not passed, this node will be host of teams that doesn't have a node associated for

them. The `-h` flag is mandatory, and should be passed in the form http://tsuruhost.com, with http (or https), otherwise it won't be accepted as valid.

It's highly important that every node is announced using the `node-add` command. The configuration `docker:servers` on tsuru.conf is deprecated and will be removed.

5.12 Download

5.12.1 Client binaries

tsuru clients are also distributed in binary version, so you can just download an executable and put them somewhere in your `PATH`.

It's important to note that all binaries are platform dependent. Currently, we provide each of them in three flavors:

1. **darwin_amd64**: This is Mac OS X, 64 bits. Make sure the command `uname -ms` prints "Darwin x86_64", otherwise this binary will not work in your system;
2. **linux_386**: This is Linux, 32 bits. Make sure the command `uname -ms` prints "Linux x86", otherwise this binary will not work in your system;
3. **linux_amd64**: This is Linux, 64 bits. Make sure the command `uname -ms` prints "Linux x86_64", otherwise this binary will not work in your system.

Below are the links to the binaries, you can just download, extract the archive and put the binary somewhere in your PATH:

darwin_amd64

- tsuru: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-darwin-amd64.tar.gz
- tsuru-admin: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-admin-darwin-amd64.tar.gz
- crane: https://s3.amazonaws.com/tsuru/dist-cmd/crane-darwin-amd64.tar.gz

linux_386

- tsuru: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-linux-386.tar.gz
- tsuru-admin: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-admin-linux-386.tar.gz
- crane: https://s3.amazonaws.com/tsuru/dist-cmd/crane-linux-386.tar.gz

linux_amd64

- tsuru: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-linux-amd64.tar.gz
- tsuru-admin: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-admin-linux-amd64.tar.gz
- crane: https://s3.amazonaws.com/tsuru/dist-cmd/crane-linux-amd64.tar.gz

5.13 tsuru Frequently Asked Questions

This document is an attempt to explain concepts you'll face when deploying and managing applications using tsuru. To request additional explanations you can open an issue on our issue tracker, talk to us at #tsuru @ freenode.net or open a thread on our mailing list.

5.13.1 What is tsuru?

tsuru is an open source polyglot cloud application platform (PaaS). With tsuru, you don't need to think about servers at all. You can write apps in the programming language of your choice, back it with add-on resources such as SQL and NoSQL databases, memcached, redis, and many others. You manage your app using the tsuru command-line tool and you deploy code using the Git revision control system, all running on the tsuru infrastructure.

5.13.2 What is an application?

An application, in tsuru, is a program's source code, dependencies list - on operational system and language level - and a Procfile with instructions on how to run that program. An application has a name, a unique address, a Platform, associated development teams, a repository and a set of units.

5.13.3 What is a unit?

A unit is an isolated Unix container or a virtual machine - depending on the configured provisioner. A unit has everything an application needs to run, the fetched operational system and language level dependencies, the application's source code, the language runtime, and the applications processes defined on the Procfile.

5.13.4 What is a platform?

A platform is a well defined pack with installed dependencies for a language or framework that a group of applications will need. A platform might be a container template, or a virtual machine image.

For instance, tsuru has a container image for python applications, with virtualenv installed and other required things needed for tsuru to deploy applications on top of that platform. Platforms are easily extendable in tsuru, but currently not managed by it, all tsuru does (by now) is to keep database records for each existent platform. Every application runs on top of a platform.

5.13.5 What is a service?

A service is a well defined API that tsuru communicates with to provide extra functionality for applications. Examples of services are MySQL, Redis, MongoDB, etc. tsuru has built-in services, but it is easy to create and add new services to tsuru. Services aren't managed by tsuru, but by its creators.

Check the *service usage documentation* for more on using services and the *building your own service tutorial* for a quick start on how to extend tsuru by creating new services.

5.13.6 How does environment variables work?

All configurations in tsuru are handled by the use of environment variables. If you need to connect with a third party service, e.g. twitter's API, you are probably going to need some extra configurations, like client_id. In tsuru, you can export those as environment variables, visible only by your application's processes.

When you bind your application into a service, most likely you'll need to communicate with that service in some way. Services can export environment variables by telling tsuru what they need, so whenever you bind your application with a service, its API can return environment variables for tsuru to export on your application's units.

5.13.7 How does the quota system works?

Quotas are handled per application and user. Every user has a quota number for applications. For example, users may have a default quota of 2 applications, so whenever a user tries to create more than two applications, he/she will receive a quota exceeded error. There are also per applications quota. This one limits the maximum number of units that an application may have.

5.13.8 How routing works?

tsuru has a router interface, which makes extremely easy to change the way routing works with any provisioner. There are two ready-to-go routers: one using hipache and another with elb.

5.13.9 How are Git repositories managed?

tsuru uses Gandalf to manage git repositories. Every time you create an application, tsuru will ask Gandalf to create a related git bare repository for you to push in.

This is the remote tsuru gives you when you create a new app. Everytime you perform a git push, Gandalf intercepts it, check if you have the required authorization to write into the application's repository, and then lets the push proceeds or returns an error message.

5.13.10 Client installation fails with "undefined: bufio.Scanner". What does it mean?

tsuru clients require Go 1.1 or later. The message `undefined: bufio.Scanner` means that you're using an old version of Go. You'll have to install the last version.

If you're using Homebrew on Mac OS, just run:

```
$ brew update
$ brew upgrade go
```

5.14 tsuru Overview

This document is in alpha state, to suggest improvements check out the related github issue.

tsuru is an open source PaaS. If you don't know what a PaaS is and what it does, see wikipedia's description.

It follows the principles described in the The Twelve-Factor App methodology.

5.14.1 Fast and easy deployment

Deploying an app is simple and easy. No special tools needed, just a plain git push. The entire process is very simple, especially from the second deployment, whether your app is big or small.

tsuru uses git as the means of deploying an application. You don't need master git in order to deploy an app to tsuru, although you will need to know the very basic workflow, add/commit/push and remote managing. Git allows really fast deploys, and tsuru makes the best possible use of it by not cloning the whole repository history of your application, there's no need to have that information in the application webserver.

tsuru will also take care of all the applications dependencies in the deployment process. You can specify operating system and language specific dependencies. For example, if you have a Python application, tsuru will search for the requirements.txt file, but first it will search for OS dependencies (a list of deb packages in a file named requirements.apt, in the case of Ubuntu).

tsuru also has *hooks* that can trigger commands before and after some events that happen during the deployment process, like restart (represented by `restart:before`, `restart:before-each`, `restart:after` and `restart:after-each` hooks).

5.14.2 Continuous Deployment

Easily create testing, staging, and production versions of your app and deploy to them instantly.

5.14.3 Add-on Resources

Instantly provision and integrate third party services with one command. tsuru provides the basic services your application will need, like searching, caching, storage and frontend; you can get all of that in a fashionable and really easy way using tsuru's command line.

5.14.4 Per-Environment Config Variables

Configuration for an application should be stored in environment variables - and we know that. tsuru lets you define your environment variables using the command line, so you can have the configuration flexibility your application need.

tsuru also makes use of environment variables. When you bind a service with your application, tsuru gives the service the ability to inject environment variables in your application environment. For instance, if you use the default MySQL service, it will inject variables for you to establish a connection with your application database.

5.14.5 Custom Services

tsuru already has services for you to use, but you don't need to use them at all if you don't want to. If you already have, let's say, a MySQL server running on your infrastructure, all you need to do in order to use it is simply configure environment variables and use them in your application config.

You can also create your own services and make them available for you and others to use it on tsuru. It's so easy to do so that you'll want to sell your own services. tsuru talks with services using a well defined API, all you have to do is implement four endpoints that knows how to provision instances of your services and bind them to tsuru applications (like creating VMs, authorizing security groups, creating ACLs, etc), and register your service in tsuru with a really simple yaml manifest.

5.14.6 Logging and Visibility

Full visibility into your app's operations with real-time logging, process status inspection, and an audit trail of all releases. tsuru will capture standard streams (output and error) from your application and expose them via the `tsuru log` command. You can also filter logs, for example, if you don't want to see the logs of developers activity (e.g.: a deploy action), you can specify the source as "app" and you'll get only the application webserver logs.

5.14.7 Process Management

tsuru manages all processes from an application, so you don't have to worry about it. But it does not know to start it. You'll have to teach tsuru how to start your application using a Procfile. tsuru reads the Procfile and uses Circus to start and manage the running process. You can even enable a web console for Circus to manage your application process and to watch CPU and memory usage in real-time through a web interface.

tsuru also allows you to easily restart your application process via command line. Although tsuru will do all the hard work of managing and fixing eventual problems with your process, you might need to restart your application manually, so we give you an easy way to do it.

5.14.8 Control Surfaces

tsuru exposes its features through a solid, stable REST API. You can write clients for this API, or you can use one of the clients maintained by tsuru developers.

tsuru ships with two API clients: the command line interface (CLI), which is pretty stable and ready for day-to-day usage; and the web interface, which is under development, but is also a great tool to manage, check logs and monitor applications and services resources.

5.14.9 Scaling

The Docker provisioner allows you to easily add and remove units, enabling one to scale an application painlessly. It will take care of the application code replication, and services binding. There's nothing required to the developer to do in order to scale an application, just add a new unit and tsuru will do the trick.

You may also want to scale using the Front end as a Service, powered by Varnish. One single application might have a whole farm of Varnish VMs in front of it handling all the traffic.

5.14.10 Built-in Database Services

tsuru already has a variety of database services available for setup on your cloud. It allows you to easily create a service instance for your application usage and bind them together. The service setup for your application is transparent by the use of environment variables, which are exported in all instances of the application, allowing your configuration to fit several environments (like development, staging, production, etc.)

5.14.11 Extensible Service and Platform Support

tsuru allows you to easily add support for new services and new platforms. For application platforms, it uses platforms based on Dockerfiles that can be dynamically added to Tsuru. See Basebuilder for more details. For services, Tsuru defines an *API* that it uses to comunicate with them.

5.14.12 Collaboration

Manage sharing and deployment of your application. tsuru uses teams to control access to resources. A developer may create a team, grant/revoke app access to/from a team or add/remove new users to/from a team. One can be a member of multiple teams and control which applications each team has access to.

5.14.13 Easy Server Deployment

tsuru itself is really easy to deploy and manage, you can get it done by following these simple steps.

5.14.14 Distributed and Extensible

tsuru server is easily extensible, distributed and customizable. It has the concept of `Provisioner`: a provisioner is a component that takes care of the orchestration (VM/container management) and provisioning. By default, it will deploy applications using the Docker provisioner, but you can easily implement your own provisioner and use whatever backend you wish.

When you extend tsuru, you are able to pratically build a new PaaS in terms of behavior of provision and orchestration, making use of the great tsuru structure. You change the whole tsuru workflow by implementing a new provisioner.

5.14.15 Dev/Ops Perspective

tsuru's components are distributed, it is composed by many pieces of software, each one made to be easily deployable and maintenable. #TODO link architecture overview.

5.14.16 Application Developer Perspective

We aim to make developers life easier. #TODO link development workflow.

5.15 Why tsuru?

This document aims to show tsuru's most killing features. Additionally, provides a comparison of tsuru with others PaaS's on the market.

5.15.1 Easy Server Installation

It's really easy to have a running PaaS with tsuru. We provide a serie of scripts, each one built to install and configure the required components for each tsuru provisioner, you can check our scripts on tsuru repository, there are separated scripts to install each component, so it's easy to create your own script to configure a new provisioner or to change the configuration of an existing one.

But it's okay if you want more control and do not want to use our scripts, or want to better understand the interaction between tsuru components, we built a guide only for you.

5.15.2 Platforms Extensibility

One of tsuru main goals is to be easily extensible. The Docker provisioner has an specific image for each platform, if one wants to create a new platform, just extend tsuru/base image and follow the directory tree structure, the scripts and Dockerfile for our existing platforms images can be found on our images repository

5.15.3 Services Creation and Extension

Most applications need a service to work properly, like a database service. tsuru provides an interface API to communicate with services APIs, but it doesn't manage services directly, this provides more control over the service and its management.

In order to create a new service you simply write an API implementing the predefined endpoints. tsuru will call when a user performs an action using the client, read more on the building your service tutorial.

You can either create a new service or modify an existing one, if its source is open. All services APIs made by tsuru team are open and contributions are very welcome. For example, the mongoDB api shares one database installation with everyone that is using it, if you don't like it and want to change it, you can do it and create a new service on tsuru with your own implementation.

5.15.4 IaaS's and Provisioners

tsuru provides an easy way to change the application unit provisioning system but right now it supports only Docker. One can simply implement the Provision interface tsuru provides, configure it on your installation and start using it.

5.15.5 Routers

tsuru also provides an abstraction for routing control and load balancing in application units. It provides a routing interface, that you can combine on many ways: you can plug any router with any provisioner, you can also create your own routing backend and plug it with any existing provisioner, this can be done only changing tsuru's configuration file.

5.15.6 Comparing tsuru With Other PaaS's

The following table compares tsuru with OpenShift and Stackato PaaS's.

If you have anything to consider, or want to ask us to add another PaaS on the list contact us in #tsuru @ freenode.net or at our mailing list

	tsuru	OpenShift	Stackato
Built-in Platforms	Node.js, PHP, HTML, Python, Ruby, Go, Java	Java, PHP, Ruby, Node.js, Python	Java, Node.Js, Perl, PHP, Python, Ruby
End-user web UI	Yes (Abyss)	Yes	Yes
CLI	Yes	Yes	Yes
Deployment hooks	Yes	No	Yes
SSH Access	Yes (management-only)	Yes	Yes
Run Commands Remotely	Yes	No	No
Application Monitoring	Yes	Yes	Yes
SQL Databases	MySQL	MySQL, PostgreSQL	MySQL, PostgreSQL
NoSQL Databases	MongoDB, Cassandra Memcached, Redis	MongoDB	MongoDB, Redis
Log Streaming	Yes	Yes (not built-in)	Yes
Metering/Billing API	No (issue 466)	No	Yes
Quota System	Yes	Yes	Yes
Container Based Apps	Yes	Yes	Yes
VMs Based Apps	Yes	No	No
Open Source	Yes	Yes	No
Free	Yes	Yes	Yes
Paid/Closed Version	No	Yes	Yes
PaaS Healing	Yes	No	No
App Healing	Yes	No	No
App Fault Tolerance	Yes	Yes (by cartridge)	Yes
Auto Scaling	No (issue 154)	Yes	Yes (for some IaaS's)
Manual Scaling	Yes	No	Yes

5.16 Deployment hooks

tsuru provides some deployment hooks, like `restart:before`, `restart:after` and `build`. Deployment hooks allow developers to run commands before and after some commands.

Hooks are listed in a special file located in the root of the application. The name of the file may be `app.yaml` or `app.yml`. Here is an example of the file:

```
hooks:
  restart:
    before:
      - python manage.py migrate
    before-each:
      - python manage.py generate_local_file
    after-each:
      - python manage.py clear_local_cache
    after:
      - python manage.py clear_redis_cache
  build:
    - python manage.py collectstatic --noinput
    - python manage.py compress
```

tsuru supports the following hooks:

- `restart:before`: this hook lists commands that will run before the app is restarted. Commands listed in this hook will run once per app.
- `restart:before-each`: this hook lists commands that will run before the unit is restarted. Commands listed in this hook will run once per unit. For instance, imagine there's an app with two units and the `app.yaml` file listed above. The command **python manage.py generate_local_file** would run two times, once per unit.
- `restart:after-each`: this hook is like before-each, but runs after restarting a unit.
- `restart:after`: this hook is like before, but runs after restarting an app.
- `build`: this hook lists commands that will be run during deploy, when the image is being generated. (only for docker provisioner)

5.17 Application Deployment

This document provides a high-level description on how application deployment works on tsuru.

5.17.1 Preparing Your Application

If you follow the 12 Factor app principles you shouldn't have to change your application in order to deploy it on tsuru. Here is what an application need to go on a tsuru cloud:

1. Well defined requirements, both, on language level and operational system level
2. Configuration of external resources using environment variables
3. A Procfile to tell how your process should be run

Let's go a little deeper through each of those topics.

1. Requirements

Every well writen application nowdays has well defined dependencies. In Python, everything is on a requirements.txt or like file, in Ruby, they go on Gemfile, Node.js has the package.json, and so on. Some of those dependencies also have operational system level dependencies, like the Nokogiri Ruby gem or MySQL-Python package, tsuru bootstraps units as clean as possible, so you also have to declare those operational system requirements you need on a file called requirements.apt. This files should have the packages declared one per-line and look like that:

```
python-dev
libmysqlclient-dev
```

2. Configuration With Environment Variables

Everything that vary between deploys (on different environments, like development or production) should be managed by environment variables. tsuru takes this principle very seriously, so all services available for usage in tsuru that requires some sort of configuration does it via environment variables so you have no pain while deploying on different environments using tsuru.

For instance, if you are going to use a database service on tsuru, like MySQL, when you bind your application into the service, tsuru will receive from the service API everything you need to connect with MySQL, e.g: user name, password, url and database name. Having this information, tsuru will export on every unit your application has the equivalent environment variables with their values. The names of those variables are defined by the service providing them, in this case, the MySQL service.

Let's take a look at the settings of tsuru hosted application built with Django:

```
import os

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": os.environ.get("MYSQLAPI_DB_NAME"),
        "USER": os.environ.get("MYSQLAPI_DB_USER"),
        "PASSWORD": os.environ.get("MYSQLAPI_DB_PASSWORD"),
        "HOST": os.environ.get("MYSQLAPI_HOST"),
        "PORT": "",
        "TEST_NAME": "test",
    }
}
```

You might be asking yourself "How am I going to know those variables names?", but don't fear! When you bind your application with tsuru, it'll return all variables the service asked tsuru to export on your application's units (without the values, since you are not gonna need them), if you lost the environments on your terminal history, again, don't fear! You can always check which service made what variables available to your application using the <insert command here>.

5.18 Building your app in tsuru

tsuru is an open source polyglot cloud application platform. With tsuru, you don't need to think about servers at all. You can write apps in the programming language of your choice, back it with add-on resources such as SQL and NoSQL databases, memcached, redis, and many others. You manage your app using the tsuru command-line tool and you deploy code using the Git revision control system, all running on the tsuru infrastructure.

5.18.1 Install the tsuru client

Install the tsuru client for your development platform.

The the tsuru client is a command-line tool for creating and managing apps. Check out the *CLI usage guide* to learn more.

5.18.2 Sign up

To create an account, you use the user-create command:

```
$ tsuru user-create youremail@domain.com
```

`user-create` will ask for your password twice.

5.18.3 Login

To login in tsuru, you use the login command, you will be asked for your password:

```
$ tsuru login youremail@domain.com
```

5.18.4 Deploy an application

Choose from the following getting started tutorials to learn how to deploy your first application using a supported language or framework:

- *Deploying Python applications in tsuru*
- *Deploying Ruby/Rails applications in tsuru*
- *Deploying PHP applications in tsuru*
- *Deploying go applications in tsuru*

5.19 Recovering an application

Your application may be downtime for a number of reasons. This page will help you discover why and what you can do to fix the problem.

5.19.1 Check your application logs

The first step is to check the application logs. To view your logs, run:

```
$ tsuru log -a appname
```

5.19.2 Restart your application

Some application issues are solved by restart. For example, your application may need to be restarted after a schema change to your database.

```
$ tsuru restart -a appname
```

5.19.3 Checking units status

```
$ tsuru app-info -a appname
```

5.20 Coding style

Please follow these coding standards when writing code for inclusion in tsuru.

5.20.1 Formatting

- Follow the go formatting style

5.20.2 Naming standards

New<Something>

is used by the <Something> *constructor*:

```
NewApp(name string) (*App, error)
```

Add<Something>

is a *method* of a type that has a collection of <Something>'s. Should receive an instance of <Something>:

```
func (a *App) AddUnit(u *Unit) error
```

Add

is a collection *method* that adds one or more elements:

```
func (a *AppList) Add( apps ...*App) error
```

Create<Something>

it's a *function* that's save an instance of <Something> in the database. Should receives an instance of <Something>.

```
func CreateApp(a *App) error
```

Delete<Something>

it's a *function* that's delete an instance of <Something> from database.

Remove<Something>

it's opposite of Add<Something>.

5.21 Setting up you tsuru development environment

To install tsuru from source, you need to have Go installed and configured. This file will guide you through the necessary steps to get tsuru's development environment.

5.21.1 Installing Go

You need to install the last version of Go to compile tsuru. You can download binaries distribution from Go website or use your preferred package installer (like Homebrew on Mac OS and apt-get on Ubuntu):

```
$ [sudo] apt-get install golang
```

```
$ brew install go
```

5.21.2 Installing MongoDB

tsuru uses MongoDB (+2.2), so you need to install it. For that, you can follow instructions on MongoDB website and download binary distributions (http://www.mongodb.org/downloads). You can also use your preferred package installer:

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
$ sudo bash -c 'echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" > /etc/
$ sudo apt-get update
$ sudo apt-get install mongodb-10gen -y

$ brew install mongodb
```

5.21.3 Installing Beanstalkd

tsuru uses Beanstalkd as a work queue. Install the latest version, by doing this:

```
$ sudo apt-get install -y beanstalkd

$ brew install beanstalkd
```

5.21.4 Installing Redis

One of tsuru routing providers uses Redis to store information about frontends and backends. You will also need to install it:

```
$ sudo apt-get install -y redis-server

$ brew install redis
```

5.21.5 Installing git, bzr and mercurial

tsuru depends on go libs that use git, bazaar and mercurial, so you need to install these three version control systems to download and compile tsuru from source.

To install git, you can use your package installer:

```
$ sudo apt-get install git

$ brew install git
```

To install bazaar, follow the instructions in bazaar's website (http://wiki.bazaar.canonical.com/Download), or use your package installer:

```
$ sudo apt-get install bzr

$ brew install bzr
```

To install mercurial, you can also follow instructions on its website (http://mercurial.selenic.com/downloads/) or use your package installer:

```
$ sudo apt-get install mercurial

$ brew install mercurial
```

5.21.6 Setting up GOPATH and cloning the project

Go uses an environment variable called GOPATH to allow users to develop using the go build tool (http://golang.org/cmd/go). So you need to setup this variable before cloning and installing tsuru. You can set this variable to your $HOME directory, or something like *$HOME/gocode*.

Once you have defined the GOPATH variable, then run the following commands:

```
$ mkdir -p $GOPATH/src/github.com/tsuru
$ cd $GOPATH/src/github.com/tsuru
$ git clone git://github.com/tsuru/tsuru
```

If you have already cloned the repository, just move the cloned directory to *$GOPATH/src/github.com/tsuru*.

Also, you will definitely want to add $GOPATH/bin to your $PATH.

For more details on GOPATH, please check this url: http://golang.org/cmd/go/#GOPATH_environment_variable

5.21.7 Starting Redis, Beanstalkd and MongoDB

Before building the code and running the tests, execute the following commands to start Redis, Beanstalkd and MongoDB processes.

```
$ redis-server
$ mongod
$ beanstalkd -l 127.0.0.1
```

5.21.8 Installing tsuru dependencies and running tests

You can use *make* to install all tsuru dependencies and run tests. It will also check if everything is ok with your GOPATH setup:

```
$ make
```

5.22 Building a tsuru development environment with Vagrant

First, make sure that virtualbox, vagrant and git are installed on your machine.

Then clone the *tsuru-bootstrap* project from github:

```
git clone https://github.com/dgryski/tsuru-bootstrap.git
```

Enter the *tsuru-bootstrap* directory and execute *vagrant up*. It will take a time:

```
cd tsuru-bootstrap
vagrant up
```

After it, configure the tsuru target with the address of the server that's running by vagrant:

```
tsuru target-add development http://192.168.50.4:8080 -s
```

Now you can create your user and deploy your apps.

5.23 Installing tsuru clients

tsuru contains three clients: `tsuru`, `tsuru-admin` and `crane`.

- **tsuru** is the command line utility used by application developers, that will allow users to create, list, bind and manage apps. For more details, check *tsuru usage*;
- **crane** is used by service administrators. For more detail, check *crane usage*;
- **tsuru-admin** is used by cloud administrators. Whoever is allowed to use it has gotten super powers :-)

This document describes how you can install those clients, using pre-compiled binaries or building them from source.

5.23.1 Using homebrew (Mac OS X only)

If you use Mac OS X and homebrew, you may use a custom tap to install `tsuru`, `crane` and `tsuru-admin`. First you need to add the tap:

```
$ brew tap tsuru/homebrew-tsuru
```

Now you can install tsuru, tsuru-admin and crane:

```
$ brew install tsuru
$ brew install tsuru-admin
$ brew install crane
```

Whenever a new version of any of tsuru's clients is out, you can just run:

```
$ brew update
$ brew upgrade <formula> # tsuru/tsuru-admin/crane
```

For more details on taps, check homebrew documentation.

NOTE: tsuru requires Go 1.1 or higher. Make sure you have the last version of Go installed in your system.

5.23.2 Using the PPA (Ubuntu only)

Ubuntu users can install tsuru clients using `apt-get` and the tsuru PPA. You'll need to add the PPA repository locally and run an `apt-get update`:

```
$ sudo apt-add-repository ppa:tsuru/ppa
$ sudo apt-get update
```

Now you can install tsuru's clients:

```
$ sudo apt-get install tsuru
$ sudo apt-get install crane
$ sudo apt-get install tsuru-admin
```

5.23.3 Using AUR (ArchLinux only)

Archlinux users can build and install tsuru client from AUR repository, Is needed to have installed yaourt program.

You can run:

```
$ yaourt -S tsuru
```

5.23.4 Pre-built binaries (Linux and Mac OS X)

tsuru clients are also distributed in binary version, so you can just download an executable and put them somewhere in your `PATH`.

It's important to note that all binaries are platform dependent. Currently, we provide each of them in three flavors:

1. **darwin_amd64**: This is Mac OS X, 64 bits. Make sure the command `uname -ms` prints "Darwin x86_64", otherwise this binary will not work in your system;
2. **linux_386**: This is Linux, 32 bits. Make sure the command `uname -ms` prints "Linux x86", otherwise this binary will not work in your system;
3. **linux_amd64**: This is Linux, 64 bits. Make sure the command `uname -ms` prints "Linux x86_64", otherwise this binary will not work in your system.

Below are the links to the binaries, you can just download, extract the archive and put the binary somewhere in your PATH:

darwin_amd64

- tsuru: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-darwin-amd64.tar.gz
- tsuru-admin: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-admin-darwin-amd64.tar.gz
- crane: https://s3.amazonaws.com/tsuru/dist-cmd/crane-darwin-amd64.tar.gz

linux_386

- tsuru: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-linux-386.tar.gz
- tsuru-admin: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-admin-linux-386.tar.gz
- crane: https://s3.amazonaws.com/tsuru/dist-cmd/crane-linux-386.tar.gz

linux_amd64

- tsuru: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-linux-amd64.tar.gz
- tsuru-admin: https://s3.amazonaws.com/tsuru/dist-cmd/tsuru-admin-linux-amd64.tar.gz
- crane: https://s3.amazonaws.com/tsuru/dist-cmd/crane-linux-amd64.tar.gz

5.23.5 Build from source (Linux and Mac OS X)

tsuru's source is written in Go, so before installing tsuru from source, please make sure you have installed and configured Go.

With Go installed and configured, you can use `go get` to install any of tsuru's clients:

```
$ go get github.com/tsuru/tsuru/cmd/tsuru
$ go get github.com/tsuru/tsuru/cmd/tsuru-admin
$ go get github.com/tsuru/tsuru/cmd/crane
```

5.24 Howto install a dns forwarder

This document describes how to create a dns forwarder and set a base domain for tsuru.

5.24.1 Overview

The recommended way to use tsuru is integrated with a DNS server. The easiest way to do that is configuring it as a cache forwarder, and configuring a DNS zone to be used for tsuru as required.

5.24.2 Installing Bind

Here you will see how easy is to install a DNS server. Bellow you will see a howto for Ubuntu and Centos

Ubuntu

```
$ apt-get install bind9 bind9utils -y
```

Centos

```
$ yum install bind bind-utils -y
$ chkconfig named on
$ service named start
```

5.24.3 Configuring Bind

Forwarder

First we will show how to configure your DNS as a forwarder. Into the config file, insert the forwarders directive inside the "options" main directive. You can use the google's public DNS(8.8.8.8/8.8.4.4) as forwarder or your company's DNS. It should look like that:

Ubuntu

```
$ egrep -v '//|^$' /etc/bind/named.conf.options
options {
        directory "/var/cache/bind";
        forwarders {
                8.8.8.8;
                8.8.4.4;
        };
        dnssec-validation auto;
        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };
};
```

Centos

```
$   egrep -v '//|^$' /etc/named.conf |head
options {
    forwarders { 8.8.8.8; 8.8.4.4; };
    listen-on port 53 { any; };
    listen-on-v6 port 53 { ::1; };
    directory           "/var/named";
    dump-file           "/var/named/data/cache_dump.db";
    statistics-file     "/var/named/data/named_stats.txt";
    memstatistics-file  "/var/named/data/named_mem_stats.txt";
    allow-query         { any; }";
    recursion yes;
```

DNS Zone

Now we will set a DNS Zone to be used by tsuru. In this example we are using the domain cloud.company.com. Create a entrance for that into /etc/bind/named.conf.local(for ubuntu) or /etc/named.conf(for centos) as following:

Ubuntu

```
zone "cloud.company.com" {
        type master;
        file "/etc/bind/db.cloud.company.com";
};
```

Centos

```
zone "cloud.company.com" {
        type master;
        file "db.cloud.company.com";
};
```

And create a db.cloud.company.com file(considering the your external IP for tsuru, hipache and git is 192.168.123.131) the way below:

```
$  cat db.cloud.company.com
;
$TTL    604800
@       IN      SOA     cloud.company.com. tsuru.cloud.company.com. (
                              3         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      cloud.company.com.
@       IN      A       192.168.123.131
git     IN      A       192.168.123.131 ; here we can set a better exhibition for the git remote prov
*       IN      A       192.168.123.131
```

Ps: If you have problems, it could be related with the date of your machine. We recommend you to install a ntpd service.

Now just reload your DNS server, point it to your resolv.conf, and use tsuru! To test, just execute the command below, and see if all responses resolv to 192.168.123.131:

```
$ ping cloud.company.com
$ ping git.cloud.company.com
$ ping zzzzz.cloud.company.com
$ ping anydomain.cloud.company.com
```

5.25 Release notes

Release notes for the official tsuru releases. Each release note will tell you what's new in each version.

5.25.1 tsr

tsr is the tsuru server daemon.

0.5.1 release

tsr 0.5.1 release notes

Welcome to tsr 0.5.1!

These release notes cover the new features, bug fixes and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.5.0 or older versions.

What's new in tsr 0.5.1

- **tsr api** now checks tsuru.conf file and refuse to start if it is misconfigured. It's also possible to exclusively test the config file with the -t flag. i.e.: running "tsr api -t". (#714).
- new command in the `tsuru-admin`: the command fix-containers will look for broken containers and fix their configuration within the router, and in the database

Bug fixes

- Do not lock application on `tsuru run`

Backwards incompatible changes

- **tsr collector** is no more. In the 0.5.0 release, collector got much less responsibilities, and now it does nothing, because it no longer exists. The last of its responsibilities is now available in the `tsuru-admin fix-containers` command.

0.5.0 release

tsr 0.5.0 release notes

Welcome to tsr 0.5.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.4.0 or older versions.

What's new in tsr 0.5.0

Stability and Consistency One of the main feature on this release is improve the stability and consitency of the tsuru API.

- prevent inconsitency caused by problems on deploy (#803) / (#804)
- units information is not updated by collector (#806)
- fixed log listener on multiple API hosts (#762)
- prevent inconsitency caused by simultaneous operations in an application (#789)
- prevent inconsitency cause by simultaneous `env-set` calls (#820)
- store information about errors and identify flawed application deployments (#816)

Buildpack tsuru now supports deploying applications using Heroku Buildpacks.

Buildpacks are useful if you're interested in following Heroku's best practices for building applications or if you are deploying an application that already runs on Heroku.

tsuru uses Buildstep Docker image to deploy applications using buildpacks. For more information, take a look at the buildpacks documentation page: http://docs.tsuru.io/en/latest/using/buildpacks.html.

Other features

- filter application logs by unit (#375)
- support for deployments with archives, which enables the use of the `pre-receive` Git hook, and also deployments without Git (#458, #442 and #701)
- stop and start commands (#606)
- oauth support (#752)
- platform update command (#780)
- support services with *https* endpoint (#812) / (#821)
- grouping nodes by pool in segregate scheduler. For more information you can see the docs about the segregate scheduler: *Schedulers*.

Platforms

- *deployment hooks* support for static and PHP applications (#607)
- new platform: buildpack (used for buildpack support)

Backwards incompatible changes

- Juju provisioner was removed. This provisioner was not being maintained. A possible idea is to use Juju in the future to provision the tsuru nodes instead of units
- ELB router was removed. This router was used only by juju.
- `tsr admin` was removed.

- The field `units` was removed from the collection `apps`. Information about units are now available in the provisioner. Now the unit state is controlled by provisioner. If you are upgrading tsuru from 0.4.0 or an older version you should run the MongoDB script bellow, where the *docker* collection name is the name configured by *docker:collection* in *tsuru.conf*:

```
var migration = function(doc) {
    doc.units.forEach(function(unit){
        db.docker.update({"id": unit.name}, {$set: {"status": unit.state}});
    });
};

db.apps.find().forEach(migration);
```

- The scheduler collection has changed to group nodes by pool. If you are using this scheduler you shoul run the MongoDB script bellow:

```
function idGenerator(id) {
    return id.replace(/\d+/g, "")
}

var migration = function(doc) {
    var id = idGenerator(doc._id);
    db.temp_scheduler_collection.update(
        {teams: doc.teams},
        {$push: {nodes: doc.address},
         $set: {teams: doc.teams, _id: id}},
        {upsert: true});
}
db.docker_scheduler.find().forEach(migration);
db.temp_scheduler_collection.renameCollection("docker_scheduler", true);
```

You can implement your own *idGenerator* to return the name for the new pools. In our case the *idGenerator* generates an id based on node name. It makes sense because we use the node name to identify a node group.

Features deprecated in 0.5.0 Beanstalkd queue backend will be removed in 0.6.0.

0.4.0 release

tsr 0.4.0 release notes

Welcome to tsr 0.4.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.x or older versions.

What's new in tsr 0.4.0

- redis queue backend was refactored.
- fixed output when service doesn't export environment variables (#772)

Docker

- refactored unit creation to be more atomic
- support for unit-agent (#633) - tsuru unit agent repository: https://github.com/tsuru/tsuru-unit-agent

- added an administrative command to move and rebalance containers between nodes (#646) - docs about rebalance: http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#containers-rebalance
- memory swap limit is configurable (#764)
- added a command to add a new platform (#780) - docs about *platform-add* command: http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#platform-add

Backwards incompatible changes The s3 integration on app creation was removed. The config properties *bucket-support*, *aws:iam aws:s3* was removed too.

You should use *tsuru* cli 0.9.0 and *tsuru-admin* 0.3.0 version.

0.3.11 release

tsr 0.3.11 release notes

Welcome to tsr 0.3.11!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.10 or older versions.

What's new in tsr 0.3.11

API

- Added app team owner - #619
- Expose public url in *create-app* - #724

Docker provisioner

- Add support to custom memory - #434

Backwards incompatible changes All existing apps have no team owner. You can run the mongodb script below to automatically set the first existing team in the app as team owner.

```
db.apps.find({ teamowner: { $exists: false }}).forEach(
    function(app) {
        app.teamowner = app.teams[0];
        db.apps.save(app);
    }
);
```

0.3.10 release

tsr 0.3.10 release notes

Welcome to tsr 0.3.10!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.9 or older versions.

What's new in tsr 0.3.10 API

- Improve feedback for duplicated users (issue #693)

Docker provisioner

- Update docker-cluster library, to fix the behavior of the default scheduler (issue #716)
- Improve debug logs for SSH (issue #665)
- Fix URL for listing containers by app

Backwards incompatible changes tsr 0.3.10 did not introduce any incompatible changes.

0.3.9 release

tsr 0.3.9 release notes

Welcome to tsr 0.3.9!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.8 or older versions.

What's new in tsr 0.3.9

API

- Login expose *is_admin* info.
- Changed get environs output data.

Backwards incompatible changes tsr 0.3.9 has changed the api output data for get environs from an app.

You should use *tsuru* cli 0.8.10 version.

0.3.8 release

tsr 0.3.8 release notes

Welcome to tsr 0.3.8!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.8 or older versions.

What's new in tsr 0.3.8

API

- Expose deploys of the app in the app-info API

Docker

- deploy hook support enviroment variables with space.

Backwards incompatible changes tsr 0.3.7 does not introduce any incompatible changes.

0.3.7 release

tsr 0.3.7 release notes

Welcome to tsr 0.3.7!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.6 or older versions.

What's new in tsr 0.3.7

API

- Improve administrative API for the Docker provisioner
- Store deploy metadata
- Improve healthcheck (ping MongoDB before marking the API is ok)
- Expose owner of the app in the app-info API

Backwards incompatible changes tsr 0.3.7 does not introduce any incompatible changes.

0.3.6 release

tsr 0.3.6 release notes

Welcome to tsr 0.3.6!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.5 or older versions.

What's new in tsr 0.3.6

Application state control

- Add new functionality to the API and provisoners: stop and starting an App

Services

- Add support for plans in services

Backwards incompatible changes tsr 0.3.6 does not introduce any incompatible changes.

0.3.5 release

tsr 0.3.5 release notes

Welcome to tsr 0.3.5!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.4 or older versions.

What's new in tsr 0.3.5

Bugfixes

- Fix administrative API for Docker provisioner

Backwards incompatible changes tsr 0.3.5 does not introduce any incompatible changes.

0.3.4 release

tsr 0.3.4 release notes

Welcome to tsr 0.3.4!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.3 or older versions.

What's new in tsr 0.3.4

Documentation improvements

- Improvements in the layout of the documentation

Bugfixes

- Swap address and cname on apps when running swap
- Always pull the image before creating the container

Backwards incompatible changes tsr 0.3.4 does not introduce any incompatible changes.

0.3.3 release

tsr 0.3.3 release notes

Welcome to tsr 0.3.3!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.2 or older versions.

What's new in tsr 0.3.3

Queue

- Add an option to use Redis instead of beanstalk for work queue

In order to use Redis, you need to change the configuration file:

```
queue: redis
redis-queue:
  host: "localhost"
  port: 6379
  db: 4
  password: "your-password"
```

All settings are optional (`queue` will still default to "beanstalkd"), refer to *configuration docs* for more details.

Other improvements and bugfixes

- Do not depend on Docker code
- Improve the layout of the documentation
- Fix multiple data races in tests
- [BUGFIX] fix bug with unit-add and application image
- [BUGFIX] fix image replication on docker nodes

Backwards incompatible changes tsr 0.3.3 does not introduce any incompatible changes.

0.3.2 release

tsr 0.3.2 release notes

Welcome to tsr 0.3.2!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.1 or older versions.

What's new in tsr 0.3.2

Segregated scheduler

- Support more than one team per scheduler
- Fix the behavior of the segregated scheduler
- Improve documentation of the scheduler

API

- Improve administrative API registration

Other improvements and bugfixes

- Do not run restart on unit-add (nor unit-remove)
- Improve node management in the Docker provisioner
- Rebuild app image on every 10 deployment

Backwards incompatible changes tsr 0.3.2 does not introduce any incompatible changes.

0.3.1 release

tsr 0.3.1 release notes

Welcome to tsr 0.3.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsuru 0.3.0 or older versions.

What's new in tsr 0.3.1

Backwards incompatible changes

0.3.0 release

tsr 0.3.0 release notes

Welcome to tsr 0.3.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsuru 0.2.x or older versions.

What's new in tsr 0.3.0

Support Docker 0.7.x and other improvements

- Fixed the 42 layers problem.
- Support all Docker storages.
- Pull image on creation if it does not exists.
- BUGFIX: when using segregatedScheduler, the provisioner fails to get the proper host address.
- BUGFIX: units losing access to services on deploy bug.

Improvements related to Services

- *bind* is atomic.
- *service-add* is atomic
- Service instance name is unique.
- Add support to bind an app without units.

Collector ticker time is configurable Now you can define the collector ticker time. To do it just set on tsuru.conf:

```
collector:
    ticker-time: 120
```

The default value is 60 seconds.

Other improvements and bugfixes

- *unit-remove* does not block util all units are removed.
- BUGFIX: send on closed channel: https://github.com/tsuru/tsuru/issues/624.
- Api handler that returns information about all deploys.
- Refactored quota backend.
- New lisp platform. Thanks to Nick Ricketts.

Backwards incompatible changes tsuru 0.3.0 handles quota in a brand new way. Users upgrading from 0.2.x need to run a migration script in the database. There are two scripts available: one for installations with quota enabled and other for installations without quota.

The easiest script is recommended for environments where quota is disabled, you'll need to run just a couple of commands in MongoDB:

```
% mongo tsuru
MongoDB shell version: x.x.x
connecting to: tsuru
> db.users.update({}, {$set: {quota: {limit: -1}}});
> db.apps.update({}, {$set: {quota: {limit: -1}}});
```

In environments where quota is enabled, the script is longer, but still simple:

```
db.quota.find().forEach(function(quota) {
    if(quota.owner.indexOf("@") > -1) {
        db.users.update({email: quota.owner}, {$set: {quota: {limit: quota.limit, inuse: quota.items
    } else {
        db.apps.update({name: quota.owner}, {$set: {quota: {limit: quota.limit, inuse: quota.items.le
    }
});
```

db.apps.update({quota: null}, {$set: {quota: {limit: -1}}}); db.users.update({quota: null}, {$set: {quota: {limit: -1}}}); db.quota.remove()

The best way to run it is saving it to a file and invoke MongoDB with the file parameter:

```
% mongo tsuru <filename.js>
```

5.25.2 tsuru

tsuru is the tsuru client.

0.10.1 release

tsuru 0.10.1 release notes

Welcome to tsuru 0.10.1.

tsuru 0.10.1 includes a bug fix in the authentication error detection when communicating with the Tsuru API.

0.10.0 release

tsuru 0.10.0 release notes

Welcome to tsuru 0.10.0!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.9.x or older versions.

What's new in tsuru 0.10.0

- added stop command.

0.9.0 release

tsuru 0.9.0 release notes

Welcome to tsuru 0.9.0!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.x or older versions.

What's new in tsuru 0.9.0

- fixed app-list output.

0.8.11 release

tsuru 0.8.11 release notes

Welcome to tsuru 0.8.11!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.11 or older versions.

What's new in tsuru 0.8.11

- new plugin system - #737

  Now is possible customize tsuru client installing and creating plugins. See the docs for more info

- app team owner is configurable - #620

  Now you can define the app team owner on *app-create*:

```
tsuru app-create appname platform -t teamname
```

0.8.10 release

tsuru 0.8.10 release notes

Welcome to tsuru 0.8.10!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.10 or older versions.

What's new in tsuru 0.8.10 Support *tsr* 0.3.9.

0.8.9.1 release

tsuru 0.8.9.1 release notes

Welcome to tsuru 0.8.9.1!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.9 or older versions.

What's new in tsuru 0.8.9.1

Improvements on app-info Now the number of deploys is displayed on *app-info* command

0.8.9 release

tsuru 0.8.9 release notes

Welcome to tsuru 0.8.9!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.8 or older versions.

What's new in tsuru 0.8.9

Improvements on app-info Now the app owner is displayed on *app-info* command

0.8.8 release

tsuru 0.8.8 release notes

Welcome to tsuru 0.8.8!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.7 or older versions.

What's new in tsuru 0.8.8

Bugfix

- Fixed a bug on *service-info* command.

0.8.7 release

tsuru 0.8.7 release notes

Welcome to tsuru 0.8.7!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.6 or older versions.

What's new in tsuru 0.8.7

Added support for service plans You can use the *service-info* command to the see the plans of a service:

```
$ tsuru service-info redis
Info for "redis"

Plans
+-------+-----------------------------------------------------------------------------------------------------------
| Name  | Description
+-------+-----------------------------------------------------------------------------------------------------------
| basic | Is a dedicated instance. With 1GB of memory.
| plus  | Is 3 dedicated instances. With 1GB of memory and HA and failover support via redis-sentinel
+-------+-----------------------------------------------------------------------------------------------------------
```

And on *service-add* the plan should be defined:

```
$ tsuru service-add redis myredis basic
Service successfully added.
```

Improvements on app-info and app-list Now the app address and the app cname is displayed on *app-info* and *app-list* command.

0.8.6 release

tsuru 0.8.6 release notes

Welcome to tsuru 0.8.6!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.5 or older versions.

What's new in tsuru 0.8.6

Improvements related to Services

- Added confirmation on *service-remove* command.

5.25.3 tsuru-admin

tsuru-admin is the tsuru administrative client.

0.4.1 release

tsuru-admin 0.4.1 release notes

Welcome to tsuru-admin 0.4.1!

tsuru-admin 0.4.1 includes a bug fix in the authentication error detection when communicating with the Tsuru API.

0.4.0 release

tsuru-admin 0.4.0 release notes

Welcome to tsuru-admin 0.4.0!

These release notes cover the new features when upgrading from tsuru-admin 0.3.x or older versions.

What's new in tsr 0.4.0 New commands:

- platform-update(#780): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#platform-update

0.3.0 release

tsuru-admin 0.3.0 release notes

Welcome to tsuru-admin 0.3.0!

These release notes cover the new features when upgrading from tsuru-admin 0.2.x or older versions.

What's new in tsr 0.3.0 New commands:

- containers-move (#756): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#containers-move
- container-move (#754): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#container-move
- containers-rebalance (#646): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#containers-rebalance
- platform-add (#283): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#platform-add

5.26 Backing up tsuru database

In the tsuru repository, you will find two useful scripts in the directory `misc/mongodb`: `backup.bash` and `healer.bash`. In this page you will learn the purpose of these scripts and how to use them.

5.26.1 Dependencies

The script `backup.bash` uses S3 to store archives, and `healer.bash` downloads archives from S3 buckets. In order to communicate with S3 API, both scripts use s3cmd.

So, before running those scripts, make sure you have installed s3cmd. You can install it using your preferred package manager. For more details, refer to its download documentation.

After installing s3cmd, you will need to configure it, by running the command:

```
$ s3cmd --configure
```

5.26.2 Saving data

The script `backup.bash` runs `mongodump`, creates a tar archive and send the archive to S3. Here is how you use it:

```
$ ./misc/mongodb/backup.bash s3://mybucket localhost database
```

The first parameter is the S3 bucket. The second parameter is the database host. You can provide just the hostname, or the host:port (for example, 127.0.0.1:27018). The third parameter is the name of the database.

5.26.3 Automatically restoring on data loss

The other script in the `misc/mongodb` directory is `healer.bash`. This script checks a list of collections and if any of them is gone, download the last three backup archives and fix all gone collections.

This is how you should use it:

```
$ ./misc/mongodb/healer.bash s3://mybucket localhost mongodb repositories users
```

The first three parameters mean the same as in the backup script. From the fourth parameter onwards, you should list the collections. In the example above, we provided two collections: “repositories” and “users”.

5.27 Server installation guide

5.27.1 Dependencies

tsuru depends on Go and libyaml.

To install Go, follow the official instructions in the language website: http://golang.org/doc/install.

To install libyaml, you can use one package manager, or download it and install it from source. To install from source, follow the instructions on PyYAML wiki: http://pyyaml.org/wiki/LibYAML.

The following instructions are system specific:

FreeBSD

```
$ cd /usr/ports/textproc/libyaml
$ make install clean
```

Mac OS X (homebrew)

```
$ brew install libyaml
```

Ubuntu

```
$ [sudo] apt-get install libyaml-dev
```

CentOS

```
$ [sudo] yum install libyaml-devel
```

5.27.2 Installation

After installing and configuring go, and installing libyaml, just run in your terminal:

```
$ go get github.com/tsuru/tsuru/...
```

5.27.3 Server configuration

TODO!

5.28 api workflow

tsuru sends requests to your service to:

- create a new instance of your service
- bind an app with your service
- unbind an app
- destroy an instance

5.28.1 Creating a new instance

This process begins when a tsuru customer creates an instance of your service via command line tool:

```
$ tsuru service-add mysql mysql_instance
```

tsuru calls your service to create a new instance of your service via POST on `/resources` (please notice that tsuru does not include a trailing slash) with the "name" that represents the app name in the request body. Example of request:

```
POST /resources HTTP/1.0
Content-Length: 19

name=mysql_instance
```

Your API should return the following HTTP response code with the respective response body:

- 201: when the instance is successfully created. You don't need to include any content in the response body.

- 500: in case of any failure in the creation process. Make sure you include an explanation for the failure in the response body.

5.28.2 Binding an app to a service instance

This process begins when a tsuru customer binds an app to an instance of your service via command line tool:

```
$ tsuru bind mysql_instance --app my_app
```

tsuru calls your service to bind an app with a service instance via POST on `/resources/<service-name>` (please notice that tsuru does not include a trailing slash) with the "hostname" that represents the app hostname in the request body. Example of request:

```
POST /resources/mysql_instance HTTP/1.0
Content-Length: 25

hostname=myapp.myhost.com
```

Your API should return the following HTTP response code with the respective response body:

- 201: if the app is successfully binded to the instance. The response body must be a JSON containing the environment variables from this instance that should be exported in the app in order to connect to the instance. If your service does not export any environment variable, write `null` or `{}` in the response body. Example of response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json; charset=UTF-8

{"MYSQL_HOST":"10.10.10.10","MYSQL_PORT":3306,"MYSQL_USER":"ROOT","MYSQL_PASSWORD":"s3cr3t","MYSQL_DA
```

Status codes for errors in the process:

- 404: if the service instance does not exist. You don't need to include any content in the response body.
- 412: if the service instance is still being provisioned, and not ready for binding yet. You can optionally include an explanation in the response body.
- 500: in case of any failure in the bind process. Make sure you include an explanation for the failure in the response body.

5.28.3 Unbind an app from a service instance

This process begins when a tsuru customer unbinds an app from an instance of your service via command line tool:

```
$ tsuru unbind mysql_instance --app my_app
```

tsuru calls your service to unbind an app with a service instance via DELETE on `/resources/<service-name>/hostname/<app-hostname>` (please notice that tsuru does not include a trailing slash). Example of request:

```
DELETE /resources/mysql_instance/hostname/myapp.myhost.com HTTP/1.0
Content-Length: 0
```

Your API should return the following HTTP response code with the respective response body:

- 200: if the app is successfully unbinded from the instance. You don't need to include any content in the response body.
- 404: if the service instance does not exist. You don't need to include any content in the response body.

- 500: in case of any failure in the unbind process. Make sure you include an explanation for the failure in the response body.

5.28.4 Destroying an instance

This process begins when a tsuru customer removes an instance of your service via command line tool:

```
$ tsuru service-remove mysql_instance
```

tsuru calls your service to remove an instance of your service via DELETE on `/resources/<service-name>` (please notice that tsuru does not include a trailing slash). Example of request:

```
DELETE /resources/mysql_instance HTTP/1.0
Content-Length: 0
```

Your API should return the following HTTP response code with the respective response body:

- 200: if the service is successfully destroyed. You don't need to include any content in the response body.
- 404: if the service instance does not exist. You don't need to include any content in the response body.
- 500: in case of any failure in the destroy process. Make sure you include an explanation for the failure in the response body.

5.28.5 Checking the status of an instance

This process begins when a tsuru customer wants to check the status of an instance via command line tool:

```
$ tsuru service-status mysql_instance
```

tsuru calls your service to check the status of the instance via GET on `/resources/mysql_instance/status` (please notice that tsuru does not include a trailing slash). Example of request:

```
GET /resources/mysql_instance/status HTTP/1.0
```

Your API should return the following HTTP response code, with the respective response body:

- 202: the instance is still being provisioned (pending). You don't need to include any content in the response body.
- 204: the instance is running and ready for connections (running). You don't need to include any content in the response body.
- 500: the instance is not running, nor ready for connections. Make sure you include the reason why the instance is not running.

5.28.6 Additional info about an instance

You can add additional info about instances of your service. To do it it's needed to implement the resource below:

```
GET /resources/mysql_instance HTTP/1.0
```

Your API should return the following HTTP response code, with the respective body:

- 404: when your api doesn't have extra info about the service instance. You don't need to include any content in the response body.
- 200: when your app has an extra info about the service instance. The response body must be a JSON containing a list of fields. A field is composed by two key/value's *label* and *value*:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

[{"label": "my label", "value": "my value"}, {"label": "myLabel2.0", "value": "my value 2.0"}]
```

5.29 Building your service

5.29.1 Overview

This document is a hands-on guide to turning your existing cloud service into a tsuru service.

In order to create a service you need to implement a provisioning API for your service, which tsuru will call using HTTP protocol when a customer creates a new instance or binds a service instance with an app.

You will also need to create a YAML document that will serve as the service manifest. We provide a command-line tool to help you to create this manifest and manage your service.

5.29.2 Creating your service api

To create your service api you can use any programming language or framework. In this tutorial we will use flask.

Prerequisites

First, let's be sure that Python and pip are already installed:

```
$ python --version
Python 2.7.2

$ pip
Usage: pip COMMAND [OPTIONS]

pip: error: You must give a command (use "pip help" to see a list of commands)
```

For more information about how to install python you can see the Python download documentation and about how to install pip you can see the pip installation instructions.

Now, with python and pip installed, you can use pip to install flask:

```
$ pip install flask
```

With flask installed let's create a file called api.py and add the code to create a minimal flask app:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

For run this app you can do:

```
$ python api.py
 * Running on http://127.0.0.1:5000/
```

If you open your web browser and access the url "http://127.0.0.1:5000/" you will see the "Hello World!".

Then, you need to implement the resources expected by the *tsuru api workflow*.

Provisioning the resource for new instances

For new instances tsuru sends a POST to /resources with the "name" that represents the service instance name in the request body. If the service instance is successfully created, your API should return 201 in status code.

Let's create a method for this action:

```
@app.route("/resources", methods=["POST"])
def add_instance():
    return "", 201
```

Implementing the bind

In the bind action, tsuru calls your service via POST on /resources/<service_name>/ with the "app-hostname" that represents the app hostname and the "unit-hostname" that represents the unit hostname on body.

If the app is successfully binded to the instance, you should return 201 as status code with the variables to be exported in the app environment on body with the json format.

As an example, let's create a method that returns a json with a fake variable called "SOMEVAR" to be injected in the app environment. To do it in flask you need to import the jsonify method.

```
from flask import jsonify

@app.route("/resources/<name>", methods=["POST"])
def bind(name):
    out = jsonify(SOMEVAR="somevalue")
    return out, 201
```

Implementing the unbinding

In the unbind action, tsuru calls your service via DELETE on /resources/<service_name>/hostname/<unit_hostname>/.

If the app is successfully unbinded from the instance you should return 200 as status code.

Let's create a method for this action:

```
@app.route("/resources/<name>/hostname/<host>", methods=["DELETE"])
def unbind(name, host):
    return "", 200
```

Implementing the destroy service instance

In the destroy action, tsuru calls your service via DELETE on /resources/<service_name>/.

If the service instance is successfully removed you should return 200 as status code.

Let's create a method for this action:

```
@app.route("/resources/<name>", methods=["DELETE"])
def remove_instance(name):
    return "", 200
```

Implementing the url for status checking

To check the status of an instance, tsuru uses the url `/resources/<service_name>/status`. If the instance is ok, this URL should return 204.

Let's create a function for this action:

```
@app.route("/resources/<name>/status", methods=["GET"])
def status(name):
    return "", 204
```

The final code for our "fake api" developed in flask is:

```
from flask import Flask
from flask import jsonify

app = Flask(__name__)


@app.route("/resources/<name>", methods=["POST"])
def bind(name):
    out = jsonify(SOMEVAR="somevalue")
    return out, 201


@app.route("/resources/<name>/hostname/<host>", methods=["DELETE"])
def unbind(name, host):
    return "", 200


@app.route("/resources", methods=["POST"])
def add_instance():
    return "", 201


@app.route("/resources/<name>", methods=["DELETE"])
def remove_instance(name, host):
    return "", 200


@app.route("/resources/<name>/status", methods=["GET"])
def status(name):
    return "", 204


if __name__ == "__main__":
    app.run()
```

5.29.3 Creating a service manifest

Using crane you can create a manifest template:

```
$ crane template
```

This will create a manifest.yaml in your current path with this content:

```
id: servicename
endpoint:
    production: production-endpoint.com
    test: test-endpoint.com:8080
```

The manifest.yaml is used by crane to defined an id and an endpoint to your service.

Change the id and the endpoint values with the information of your service:

```
id: fakeserviceid1
endpoint:
    production: fakeserviceid1.com
```

5.29.4 Submiting your service

To submit your service, you can run:

```
$ crane create manifest.yaml
```

5.30 HOWTO Install a MySQL service

First, you must have a MariaDB server, the best "mysql" server in the market. You can also use the standard mysql-server.

```
# Ubuntu 13.04
$ sudo apt-get install software-properties-common
$ sudo gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys CBCB082A1BB943DB
$ sudo gpg -a --export CBCB082A1BB943DB | sudo apt-key add -
$ sudo add-apt-repository 'deb http://mirror.aarnet.edu.au/pub/MariaDB/repo/10.0/ubuntu raring main'
$ sudo apt-get update
$ sudo apt-get install mariadb-server

# Centos - creating the mariadb repository
$ cat > /etc/yum.repos.d/MariaDB.repo <<END
# MariaDB 10.0 CentOS repository list - created 2013-09-13 13:25 UTC
# http://mariadb.org/mariadb/repositories/
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.0/centos6-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
END
$ rpm --import https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
$ yum install MariaDB-server MariaDB-client
$ service mysql start
$ chkconfig mysql on
```

After that, all you need is to create a database admin user for this service, with all necessary grants

```
# Creating a database user(log into the database with the root user)
> GRANT ALL PRIVILEGES ON *.* TO 'tsuru'@'%' IDENTIFIED BY 'password' with GRANT OPTION;
> FLUSH PRIVILEGES;
```

Now, you will install our mysql-api service example. Just create an application that will be responsible for this service

```
# Create a database for this service (change the 192.168.123.131 for your mysql server host)
$ echo "CREATE DATABASE mysqlapi" | mysql -h 192.168.123.131 -u tsuru -ppassword
# In a machine with tsuru client and crane installed
$ git clone https://github.com/globocom/mysqlapi
# Create the mysqlapi application using python as its platform.
$ tsuru app-create mysql-api python
```

In order to have mysql API ready to receive requests, we need some bootstrap stuff.

```
#First export the django settings variable:
$ tsuru env-set --app mysql-api DJANGO_SETTINGS_MODULE=mysqlapi.settings
# Inject the right environment for that service
$ tsuru env-set -a mysql-api MYSQLAPI_DB_NAME=mysqlapi
$ tsuru env-set -a mysql-api MYSQLAPI_DB_USER=tsuru
$ tsuru env-set -a mysql-api MYSQLAPI_DB_PASSWORD=password
$ tsuru env-set -a mysql-api MYSQLAPI_DB_HOST=192.168.123.131
# To show the application's repository
$ tsuru app-info -a mysql-api|grep Repository
Repository: git@192.168.123.131:mysql-api.git
$ git push git@192.168.123.131:mysql-api.git master
#Now gunicorn is able to run with our wsgi.py configuration. After that, we need to run syncdb:
$ tsuru run --app mysql-api -- python manage.py syncdb --noinput
```

To run the API in shared mode, follow this steps

```
# First export the needed variables:
# If the shared mysql database is installed in the same vm that the app is, you can use localhost fo
$ tsuru env-set --app mysql-api MYSQLAPI_SHARED_SERVER=192.168.123.131
# Here you'll also need to set up a externally accessible endpoint to be used by the apps that are u
$ tsuru env-set --app mysql-api MYSQLAPI_SHARED_SERVER_PUBLIC_HOST=192.168.123.131
# Here the mysql user to manage the shared databases
$ tsuru env-set -a mysql-api MYSQLAPI_SHARED_USER=tsuru
$ tsuru env-set -a mysql-api MYSQLAPI_SHARED_PASSWORD=password
```

More information about the ways you can work with that api you can found here.

Now you should have your application working. You just need to submit the mysqlapi service via crane. The manifest.yaml is used by crane to define an id and an endpoint to your service. For more details, see the text "Services API Workflow": http://docs.tsuru.io/en/latest/services/api.html To submit your new service, you can run:

```
# Configure the service template and point it to the application service (considering that your doma
$  cat manifest.yaml
id: mysqlapi
   endpoint:
   production: mysql-api.cloud.company.com
$ crane create manifest.yaml
```

To list your services:

```
$ crane list
#OR
$ tsuru service-list
```

This will return something like:

```
+----------+-----------+
| Services | Instances |
+----------+-----------+
```

```
| mysqlapi |           |
+----------+-----------+
```

It would be nice if your service had some documentation. To add a documentation to you service you can use:

```
$ crane doc-add mysqlapi doc.txt
```

Crane will read the content of the file and save it.

To show the current documentation of your service:

```
$ crane doc-get mysqlapi
```

doc-get will retrieve the current documentation of the service.

5.30.1 Further instructions

Now you can add this service for your applications using the bind command

For a complete reference, check the documentation for crane command: http://godoc.org/github.com/tsuru/tsuru/cmd/crane.

5.31 Crane usage

First, you must set the target with your server url, like:

```
$ crane target tsuru.myhost.com
```

After that, all you need is to create a user and authenticate:

```
$ crane user-create youremail@gmail.com
$ crane login youremail@gmail.com
```

To generate a service template:

```
$ crane template
```

This will create a manifest.yaml in your current path with this content:

```
id: servicename
endpoint:
  production: production-endpoint.com
    test: test-endpoint.com:8080
```

The manifest.yaml is used by crane to define an id and an endpoint to your service.

To submit your new service, you can run:

```
$ crane create path/to/your/manifest.yaml
```

To list your services:

```
$ crane list
```

This will return something like:

```
+----------+-----------+
| Services | Instances |
+----------+-----------+
| mysql    | my_db     |
+----------+-----------+
```

To update a service manifest:

```
$ crane create path/to/your/manifest.yaml
```

To remove a service:

```
$ crane remove service_name
```

It would be nice if your service had some documentation. To add a documentation to you service you can use:

```
$ crane doc-add service_name path/to/your/docfile
```

Crane will read the content of the file and save it.

To show the current documentation of your service:

```
$ crane doc-get service_name
```

5.31.1 Further instructions

For a complete reference, check the documentation for crane command: http://godoc.org/github.com/tsuru/tsuru/cmd/crane.

5.32 Using Buildpacks

tsuru supports deploying applications via Heroku Buildpacks.

Buildpacks are useful if you're interested in following Heroku's best practices for building applications or if you are deploying an application that already runs on Heroku.

tsuru uses Buildstep Docker image to make deploy using buildpacks possible.

5.32.1 Creating an Application

What do you need is create an application using *buildpack* platform:

```
$ tsuru app-create myapp buildpack
```

5.32.2 Deploying your Application

Use git push master to deploy your application.

```
$ git push <REMOTE-URL> master
```

5.32.3 Included Buildpacks

A number of buildpacks come bundled by default:

- https://github.com/heroku/heroku-buildpack-ruby.git
- https://github.com/heroku/heroku-buildpack-nodejs.git
- https://github.com/heroku/heroku-buildpack-java.git
- https://github.com/heroku/heroku-buildpack-play.git
- https://github.com/heroku/heroku-buildpack-python.git
- https://github.com/heroku/heroku-buildpack-scala.git
- https://github.com/heroku/heroku-buildpack-clojure.git
- https://github.com/heroku/heroku-buildpack-gradle.git
- https://github.com/heroku/heroku-buildpack-grails.git
- https://github.com/CHH/heroku-buildpack-php.git
- https://github.com/kr/heroku-buildpack-go.git
- https://github.com/oortcloud/heroku-buildpack-meteorite.git
- https://github.com/miyagawa/heroku-buildpack-perl.git
- https://github.com/igrigorik/heroku-buildpack-dart.git
- https://github.com/rhy-jot/buildpack-nginx.git
- https://github.com/Kloadut/heroku-buildpack-static-apache.git
- https://github.com/bacongobbler/heroku-buildpack-jekyll.git
- https://github.com/ddollar/heroku-buildpack-multi.git

tsuru will cycle through the bin/detect script of each buildpack to match the code you are pushing.

5.32.4 Using a Custom Buildpack

To use a custom buildpack, set the BUILDPACK_URL environment variable.

```
$ tsuru env-set BUILDPACK_URL=https://github.com/dpiddy/heroku-buildpack-ruby-minimal
```

On your next git push, the custom buildpack will be used.

5.32.5 Creating your own Buildpack

You can follow this Heroku documentation to learn how to create your own Buildpack: https://devcenter.heroku.com/articles/buildpack-api.

5.33 Procfile

Procfile is a simple text file called *Procfile* that describe the components required to run an applications. It is the way to tell to *tsuru* how to run your applications.

This document describes some of the more advances features of and the Procfile ecosystem.

A *Procfile* should look like:

```
web: gunicorn -w 3 wsgi
```

5.33.1 Syntax

Procfile is a plain text file called *Procfile* placed at the root of your application.

Each project should be represented by a name and a command, like bellow:

```
<name>: <command>
```

The *name* is a string which may contain alphanumerics and underscores and identifies one type of process.

command is a shell commandline which will be executed to spawn a process.

5.33.2 Environment variables

You can reference yours environment variables in the command:

```
web: ./manage.py runserver 0.0.0.0:$PORT
```

For more information about *Procfile* you can see the honcho documentation about *Procfile*: https://honcho.readthedocs.org/en/latest/ .

5.34 unit states

5.34.1 pending

Is when the unit is waiting to be provisioned by the tsuru provisioner.

5.34.2 bulding

Is while the unit is provisioned, it's occurs while a deploy.

5.34.3 error

Is when the an error occurs caused by the application code.

5.34.4 down

Is when an error occurs caused by tsuru internal problems.

5.34.5 unreachable

Is when the app process is up but it is not binded in the right host ("0.0.0.0") and right port ($PORT). If your process is a worker it's state will be *unreachable*.

5.34.6 started

Is when the app process is up binded in the right host ("0.0.0.0") and right port ($PORT).

5.35 Services

You can manage your services using the tsuru command-line interface.

To list all services avaliable you can use, you can use the service-list command:

```
$ tsuru service-list
```

To add a new instance of a service, use the service-add command:

```
$ tsuru service-add <service_name> <service_instance_name>
```

To remove an instance of a service, use the service-remove command:

```
$ tsuru service-remove <service_instance_name>
```

To bind a service instance with an app you can use the bind command. If this service has any variable to be used by your app, tsuru will inject this variables in the app's environment.

```
$ tsuru bind <service_instance_name> [--app appname]
```

And to unbind, use unbind command:

```
$ tsuru unbind <service_instance_name> [--app appname]
```

For more details on the `--app` flag, see "Guessing app names" section of tsuru command documentation.

5.36 Client usage

After *installing the tsuru client*, you must set the target with the tsuru server URL, something like:

5.36.1 Setting a target

```
$ tsuru target-add default https://cloud.tsuru.io
$ tsuru target-set default
```

5.36.2 Authentication

After that, all you need is to create a user and authenticate to start creating apps and pushing code to them. Use create-user and login:

```
$ tsuru user-create youremail@gmail.com
$ tsuru login youremail@gmail.com
```

5.36.3 Apps

Associating your user to a team

You need to be member of a team to create an app. To create a new team, use create-team:

```
$ tsuru team-create teamname
```

Creating an app

To create an app, use app-create:

```
$ tsuru app-create myblog <platform>
```

This will return your app's remote url, you should add it to your git repository:

```
$ git remote add tsuru git@tsuru.myhost.com:myblog.git
```

Listing your apps

When your app is ready, you can push to it. To check whether it is ready or not, you can use app-list:

```
$ tsuru app-list
```

This will return something like:

```
+-------------+-------------------------+-------------------------------------------+
| Application | Units State Summary     | Ip                                        |
+-------------+-------------------------+-------------------------------------------+
| myblog      | 1 of 1 units in-service | myblog-838381.us-east-1-elb.amazonaws.com |
+-------------+-------------------------+-------------------------------------------+
```

Showing app info

You can also use the app-info command to view information of an app. Including the status of the app:

```
$ tsuru app-info
```

This will return something like:

```
Application: myblog
Platform: gunicorn
Repository: git@githost.com:myblog.git
Teams: team1, team2
Units:
+----------+---------+
| Unit     | State   |
+----------+---------+
| myblog/0 | started |
| myblog/1 | started |
+----------+---------+
```

tsuru uses information from git configuration to guess the name of the app, for more details, see "Guessing app names" section of tsuru command documentation.

5.36.4 Public Keys

You can try to push now, but you'll get a permission error, because you haven't pushed your key yet.

```
$ tsuru key-add
```

This will search for a *id_rsa.pub* file in ~/.ssh/, if you don't have a generated key yet, you should generate one before running this command.

If you have a public key in other format (for example, DSA), you can also give the public key file to key-add:

```
$ tsuru key-add $HOME/.ssh/id_dsa.pub
```

After your key is added, you can push your application to your cloud:

```
$ git push tsuru master
```

5.36.5 Running commands

After that, you can check your app's url in the browser and see your app there. You'll probably need to run migrations or other deploy related commands. To run a single command, you should use the command run:

```
$ tsuru run "python manage.py syncdb && python manage.py migrate"
```

5.36.6 Further instructions

For a complete reference, check the documentation for tsuru command: http://godoc.org/github.com/tsuru/tsuru/cmd/tsuru.

5.37 Deploying Go applications in tsuru

5.37.1 Overview

This document is a hands-on guide to deploying a simple Go web application in tsuru.

5.37.2 Creating the app within tsuru

To create an app, you use app-create command:

```
$ tsuru app-create <app-name> <app-platform>
```

For go, the app platform is, guess what, `go`! Let's be over creative and develop a hello world tutorial-app, let's call it "helloworld":

```
$ tsuru app-create helloworld go
```

To list all available platforms, use platform-list command.

You can see all your applications using app-list command:

```
$ tsuru app-list
+-------------+-------------------------+---------+--------+
| Application | Units State Summary     | Address | Ready? |
+-------------+-------------------------+---------+--------+
| helloworld  | 0 of 0 units in-service |         | No     |
+-------------+-------------------------+---------+--------+
```

Once your app is ready, you will be able to deploy your code, e.g.:

```
$ tsuru app-list
+-------------+-------------------------+-------------+--------+
| Application | Units State Summary     | Address     | Ready? |
+-------------+-------------------------+-------------+--------+
| helloworld  | 0 of 0 units in-service |             | Yes    |
+-------------+-------------------------+-------------+--------+
```

5.37.3 Application code

A simple web application in go *main.go*:

```
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", hello)
    fmt.Println("listening...")
    err := http.ListenAndServe(":8888", nil)
    if err != nil {
        panic(err)
    }
}

func hello(res http.ResponseWriter, req *http.Request) {
    fmt.Fprintln(res, "hello, world")
}
```

5.37.4 Git deployment

When you create a new app, tsuru will display the Git remote that you should use. You can always get it using app-info command:

```
$ tsuru app-info --app blog
Application: go
Repository: git@cloud.tsuru.io:blog.git
Platform: go
Teams: myteam
Address:
```

The git remote will be used to deploy your application using git. You can just push to tsuru remote and your project will be deployed:

```
$  git push git@cloud.tsuru.io:helloworld.git master
Counting objects: 86, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (75/75), done.
Writing objects: 100% (86/86), 29.75 KiB, done.
Total 86 (delta 2), reused 0 (delta 0)
remote: Cloning into '/home/application/current'...
remote: requirements.apt not found.
remote: Skipping...
remote: /home/application/current /
####################################
#          OMIT (see below)        #
####################################
remote:  ---> App will be restarted, please check its log for more details...
remote:
To git@cloud.tsuru.io:helloworld.git
* [new branch]      master -> master
```

If you get a "Permission denied (publickey).", make sure you're member of a team and have a public key added to tsuru. To add a key, use key-add command:

```
$ tsuru key-add ~/.ssh/id_rsa.pub
```

You can use `git remote add` to avoid typing the entire remote url every time you want to push:

```
$ git remote add tsuru git@cloud.tsuru.io:helloworld.git
```

Then you can run:

```
$ git push tsuru master
Everything up-to-date
```

And you will be also able to omit the `--app` flag from now on:

```
$ tsuru app-info
Application: helloworld
Repository: git@cloud.tsuru.io:blog.git
Platform: go
Teams: myteam
Address: helloworld.cloud.tsuru.io
Units:
+--------------+---------+
| Unit         | State   |
+--------------+---------+
| 9e70748f4f25 | started |
+--------------+---------+
```

For more details on the `--app` flag, see "Guessing app names" section of tsuru command documentation.

5.37.5 Running the application

As you can see, in the deploy output there is a step described as "Restarting your app". In this step, tsuru will restart your app if it's running, or start it if it's not. But how does tsuru start an application? That's very simple, it uses a Procfile (a concept stolen from Foreman). In this Procfile, you describe how your application should be started. Here is how the Procfile should look like:

```
web: ./main
```

Now we commit the file and push the changes to tsuru git server, running another deploy:

```
$ git add Procfile
$ git commit -m "Procfile: added file"
$ git push tsuru master
####################################
#                OMIT              #
####################################
remote:  ---> App will be restarted, please check its log for more details...
remote:
To git@cloud.tsuru.io:helloworld.git
d67c3cd..f2a5d2d  master -> master
```

Now that the app is deployed, you can access it from your browser, getting the IP or host listed in `app-list` and opening it. For example, in the list below:

```
$ tsuru app-list
+-------------+-------------------------+---------------------+--------+
| Application | Units State Summary     | Address             | Ready? |
+-------------+-------------------------+---------------------+--------+
| helloworld  | 1 of 1 units in-service | blog.cloud.tsuru.io | Yes    |
+-------------+-------------------------+---------------------+--------+
```

It's done! Now we have a simple go project deployed on tsuru.

Now we can access your *app* in the URL http://helloworld.cloud.tsuru.io/.

5.37.6 Going further

For more information, you can dig into tsuru docs, or read complete instructions of use for the tsuru command.

5.38 Deploying PHP applications in tsuru

5.38.1 Overview

This document is a hands-on guide to deploying a simple PHP application in tsuru. The example application will be a very simple Wordpress project associated to a MySQL service. It's applicable to any php over apache application.

5.38.2 Creating the app within tsuru

To create an app, you use app-create command:

```
$ tsuru app-create <app-name> <app-platform>
```

For PHP, the app platform is, guess what, `php`! Let's be over creative and develop a never-developed tutorial-app: a blog, and its name will also be very creative, let's call it "blog":

```
$ tsuru app-create blog php
```

To list all available platforms, use platform-list command.

You can see all your applications using app-list command:

```
$ tsuru app-list
+-------------+-------------------------+---------+--------+
| Application | Units State Summary     | Address | Ready? |
+-------------+-------------------------+---------+--------+
| blog        | 0 of 0 units in-service |         | No     |
+-------------+-------------------------+---------+--------+
```

Once your app is ready, you will be able to deploy your code, e.g.:

```
$ tsuru app-list
+-------------+-------------------------+-------------+--------+
| Application | Units State Summary     | Address     | Ready? |
+-------------+-------------------------+-------------+--------+
| blog        | 0 of 1 units in-service |             | Yes    |
+-------------+-------------------------+-------------+--------+
```

5.38.3 Application code

This document will not focus on how to write a php blog, you can download the entire source direct from wordpress: http://wordpress.org/latest.zip. Here is all you need to do with your project:

```
# Download and unpack wordpress
$ wget http://wordpress.org/latest.zip
$ unzip latest.zip
# Preparing wordpress for tsuru
$ cd wordpress
# Notify tsuru about the necessary packages
$ echo php5-mysql > requirements.apt
# Preparing the application to receive the tsuru environment related to the mysql service
$ sed "s/'database_name_here'/getenv('MYSQL_DATABASE_NAME')/; \
            s/'username_here'/getenv('MYSQL_USER')/; \
            s/'localhost'/getenv('MYSQL_HOST')/; \
            s/'password_here'/getenv('MYSQL_PASSWORD')/" \
            wp-config-sample.php  > wp-config.php
# Creating a local git repository
$ git init
$ git add .
$ git commit -m 'initial project version'
```

5.38.4 Git deployment

When you create a new app, tsuru will display the Git remote that you should use. You can always get it using app-info command:

```
$ tsuru app-info --app blog
Application: blog
Repository: git@git.tsuru.io:blog.git
Platform: php
Teams: tsuruteam
Address:
```

The git remote will be used to deploy your application using git. You can just push to tsuru remote and your project will be deployed:

```
$ git push git@git.tsuru.io:blog.git master
Counting objects: 119, done.
```

```
Delta compression using up to 4 threads.
Compressing objects: 100% (53/53), done.
Writing objects: 100% (119/119), 16.24 KiB, done.
Total 119 (delta 55), reused 119 (delta 55)
remote:
remote:  ---> tsuru receiving push
remote:
remote: From git://cloud.tsuru.io/blog.git
remote:  * branch            master     -> FETCH_HEAD
remote:
remote:  ---> Installing dependencies
####################################
#          OMIT (see below)        #
####################################
remote:  ---> Restarting your app
remote:
remote:  ---> Deploy done!
remote:
To git@git.tsuru.io:blog.git
   a211fba..bbf5b53  master -> master
```

If you get a "Permission denied (publickey).", make sure you're member of a team and have a public key added to tsuru. To add a key, use key-add command:

```
$ tsuru key-add ~/.ssh/id_dsa.pub
```

You can use `git remote add` to avoid typing the entire remote url every time you want to push:

```
$ git remote add tsuru git@git.tsuru.io:blog.git
```

Then you can run:

```
$ git push tsuru master
Everything up-to-date
```

And you will be also able to omit the `--app` flag from now on:

```
$ tsuru app-info
Application: blog
Repository: git@git.tsuru.io:blog.git
Platform: php
Teams: tsuruteam
Address: blog.cloud.tsuru.io
Units:
+--------------+---------+
| Unit         | State   |
+--------------+---------+
| 9e70748f4f25 | started |
+--------------+---------+
```

For more details on the `--app` flag, see "Guessing app names" section of tsuru command documentation.

5.38.5 Listing dependencies

In the last section we omitted the dependencies step of deploy. In tsuru, an application can have two kinds of dependencies:

- **Operating system dependencies**, represented by packages in the package manager of the underlying operating system (e.g.: `yum` and `apt-get`);

- **Platform dependencies**, represented by packages in the package manager of the platform/language (e.g. in Python, `pip`).

All `apt-get` dependencies must be specified in a `requirements.apt` file, located in the root of your application, and pip dependencies must be located in a file called `requirements.txt`, also in the root of the application. Since we will use MySQL with PHP, we need to install the package depends on just one `apt-get` package: `php5-mysql`, so here is how `requirements.apt` looks like:

```
php5-mysql
```

You can see the complete output of installing these dependencies bellow:

```
% git push tsuru master
####################################
#                OMIT              #
####################################
Counting objects: 1155, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (1124/1124), done.
Writing objects: 100% (1155/1155), 4.01 MiB | 327 KiB/s, done.
Total 1155 (delta 65), reused 0 (delta 0)
remote: Cloning into '/home/application/current'...
remote: Reading package lists...
remote: Building dependency tree...
remote: Reading state information...
remote: The following extra packages will be installed:
remote:   libmysqlclient18 mysql-common
remote: The following NEW packages will be installed:
remote:   libmysqlclient18 mysql-common php5-mysql
remote: 0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
remote: Need to get 1042 kB of archives.
remote: After this operation, 3928 kB of additional disk space will be used.
remote: Get:1 http://archive.ubuntu.com/ubuntu/ quantal/main mysql-common all 5.5.27-0ubuntu2 [13.7 k
remote: Get:2 http://archive.ubuntu.com/ubuntu/ quantal/main libmysqlclient18 amd64 5.5.27-0ubuntu2
remote: Get:3 http://archive.ubuntu.com/ubuntu/ quantal/main php5-mysql amd64 5.4.6-1ubuntu1 [79.0 kB
remote: Fetched 1042 kB in 1s (739 kB/s)
remote: Selecting previously unselected package mysql-common.
remote: (Reading database ... 23874 files and directories currently installed.)
remote: Unpacking mysql-common (from .../mysql-common_5.5.27-0ubuntu2_all.deb) ...
remote: Selecting previously unselected package libmysqlclient18:amd64.
remote: Unpacking libmysqlclient18:amd64 (from .../libmysqlclient18_5.5.27-0ubuntu2_amd64.deb) ...
remote: Selecting previously unselected package php5-mysql.
remote: Unpacking php5-mysql (from .../php5-mysql_5.4.6-1ubuntu1_amd64.deb) ...
remote: Processing triggers for libapache2-mod-php5 ...
remote:  * Reloading web server config
remote:    ...done.
remote: Setting up mysql-common (5.5.27-0ubuntu2) ...
remote: Setting up libmysqlclient18:amd64 (5.5.27-0ubuntu2) ...
remote: Setting up php5-mysql (5.4.6-1ubuntu1) ...
remote: Processing triggers for libc-bin ...
remote: ldconfig deferred processing now taking place
remote: Processing triggers for libapache2-mod-php5 ...
remote:  * Reloading web server config
remote:    ...done.
remote: sudo: unable to resolve host 8cf20f4da877
remote: sudo: unable to resolve host 8cf20f4da877
remote: debconf: unable to initialize frontend: Dialog
remote: debconf: (Dialog frontend will not work on a dumb terminal, an emacs shell buffer, or without
remote: debconf: falling back to frontend: Readline
```

```
remote: debconf: unable to initialize frontend: Dialog
remote: debconf: (Dialog frontend will not work on a dumb terminal, an emacs shell buffer, or without
remote: debconf: falling back to frontend: Readline
remote:
remote: Creating config file /etc/php5/mods-available/mysql.ini with new version
remote: debconf: unable to initialize frontend: Dialog
remote: debconf: (Dialog frontend will not work on a dumb terminal, an emacs shell buffer, or without
remote: debconf: falling back to frontend: Readline
remote:
remote: Creating config file /etc/php5/mods-available/mysqli.ini with new version
remote: debconf: unable to initialize frontend: Dialog
remote: debconf: (Dialog frontend will not work on a dumb terminal, an emacs shell buffer, or without
remote: debconf: falling back to frontend: Readline
remote:
remote: Creating config file /etc/php5/mods-available/pdo_mysql.ini with new version
remote:
remote:  ---> App will be restarted, please check its log for more details...
remote:
To git@git.tsuru.io:ingress.git
 * [new branch]      master -> master
```

5.38.6 Running the application

As you can see, in the deploy output there is a step described as "App will be restarted". In this step, tsuru will restart your app if it's running, or start it if it's not. Now that the app is deployed, you can access it from your browser, getting the IP or host listed in `app-list` and opening it. For example, in the list below:

```
$ tsuru app-list
+-------------+-------------------------+---------------------+--------+
| Application | Units State Summary     | Address             | Ready? |
+-------------+-------------------------+---------------------+--------+
| blog        | 1 of 1 units in-service | blog.cloud.tsuru.io | Yes    |
+-------------+-------------------------+---------------------+--------+
```

5.38.7 Using services

Now that php is running, we can accesss the application in the browser, but we get a database connection error: *"Error establishing a database connection"*. This error means that we can't connect to MySQL. That's because we should not connect to MySQL on localhost, we must use a service. The service workflow can be resumed to two steps:

1. Create a service instance
2. Bind the service instance to the app

But how can I see what services are available? Easy! Use service-list command:

```
$ tsuru service-list
+----------------+-----------+
| Services       | Instances |
+----------------+-----------+
| mongodb        |           |
| mysql          |           |
+----------------+-----------+
```

The output from `service-list` above says that there are two available services: "elastic-search" and "mysql", and no instances. To create our MySQL instance, we should run the service-add command:

```
$ tsuru service-add mysql blogsql
Service successfully added.
```

Now, if we run `service-list` again, we will see our new service instance in the list:

```
$ tsuru service-list
+----------------+-----------+
| Services       | Instances |
+----------------+-----------+
| elastic-search |           |
| mysql          | blogsql   |
+----------------+-----------+
```

To bind the service instance to the application, we use the bind command:

```
$ tsuru bind blogsql
Instance blogsql is now bound to the app blog.

The following environment variables are now available for use in your app:

- MYSQL_PORT
- MYSQL_PASSWORD
- MYSQL_USER
- MYSQL_HOST
- MYSQL_DATABASE_NAME

For more details, please check the documentation for the service, using service-doc command.
```

As you can see from bind output, we use environment variables to connect to the MySQL server. Next step would be update the `wp-config.php` to use these variables to connect in the database:

```
$ grep getenv wp-config.php
define('DB_NAME', getenv('MYSQL_DATABASE_NAME'));
define('DB_USER', getenv('MYSQL_USER'));
define('DB_PASSWORD', getenv('MYSQL_PASSWORD'));
define('DB_HOST', getenv('MYSQL_HOST'));
```

You can extend your wordpress installing plugins into your repository. In the example bellow, we are adding the Amazon S3 capability to wordpress, just installing 2 more plugins: Amazon S3 and Cloudfront + Amazon Web Services. It's the right way to store content files into tsuru.

```
$ cd wp-content/plugins/
$ wget http://downloads.wordpress.org/plugin/amazon-web-services.0.1.zip
$ wget http://downloads.wordpress.org/plugin/amazon-s3-and-cloudfront.0.6.1.zip
$ unzip amazon-web-services.0.1.zip
$ unzip amazon-s3-and-cloudfront.0.6.1.zip
$ rm -f amazon-web-services.0.1.zip amazon-s3-and-cloudfront.0.6.1.zip
$ git add amazon-web-services/ amazon-s3-and-cloudfront/
```

Now you need to add the amazon AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environments support into wp-config.php. You could add these environments right after the WP_DEBUG as bellow:

```
$ grep -A2 define.*WP_DEBUG  wp-config.php
define('WP_DEBUG', false);
define('AWS_ACCESS_KEY_ID', getenv('AWS_ACCESS_KEY_ID'));
define('AWS_SECRET_ACCESS_KEY', getenv('AWS_SECRET_ACCESS_KEY'));
$ git add wp-config.php
$ git commit -m 'adding plugins for S3'
$ git push tsuru master
```

Now, just inject the right values for these environments with tsuru env-set as bellow:

```
$ tsuru env-set AWS_ACCESS_KEY_ID="xxx" AWS_SECRET_ACCESS_KEY="xxxxx" -a blog
```

It's done! Now we have a PHP project deployed on tsuru, with S3 support using a MySQL service.

5.38.8 Going further

For more information, you can dig into tsuru docs, or read complete instructions of use for the tsuru command.

5.39 Deploying Python applications in tsuru

5.39.1 Overview

This document is a hands-on guide to deploying a simple Python application in tsuru. The example application will be a very simple Django project associated to a MySQL service. It's applicable to any WSGI application.

5.39.2 Creating the app within tsuru

To create an app, you use app-create command:

```
$ tsuru app-create <app-name> <app-platform>
```

For Python, the app platform is, guess what, `python`! Let's be over creative and develop a never-developed tutorial-app: a blog, and its name will also be very creative, let's call it "blog":

```
$ tsuru app-create blog python
```

To list all available platforms, use platform-list command.

You can see all your applications using app-list command:

```
$ tsuru app-list
+-------------+-------------------------+---------+--------+
| Application | Units State Summary     | Address | Ready? |
+-------------+-------------------------+---------+--------+
| blog        | 0 of 0 units in-service |         | No     |
+-------------+-------------------------+---------+--------+
```

Once your app is ready, you will be able to deploy your code, e.g.:

```
$ tsuru app-list
+-------------+-------------------------+-------------+--------+
| Application | Units State Summary     | Address     | Ready? |
+-------------+-------------------------+-------------+--------+
| blog        | 0 of 1 units in-service |             | Yes    |
+-------------+-------------------------+-------------+--------+
```

5.39.3 Application code

This document will not focus on how to write a Django blog, you can clone the entire source direct from GitHub: https://github.com/tsuru/tsuru-django-sample. Here is what we did for the project:

1. Create the project (`django-admin.py startproject`)

2. Enable django-admin
3. Install South
4. Create a "posts" app (`django-admin.py startapp posts`)
5. Add a "Post" model to the app
6. Register the model in django-admin
7. Generate the migration using South

5.39.4 Git deployment

When you create a new app, tsuru will display the Git remote that you should use. You can always get it using app-info command:

```
$ tsuru app-info --app blog
Application: blog
Repository: git@git.tsuru.io:blog.git
Platform: python
Teams: tsuruteam
Address:
```

The git remote will be used to deploy your application using git. You can just push to tsuru remote and your project will be deployed:

```
$ git push git@git.tsuru.io:blog.git master
Counting objects: 119, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (53/53), done.
Writing objects: 100% (119/119), 16.24 KiB, done.
Total 119 (delta 55), reused 119 (delta 55)
remote:
remote:  ---> tsuru receiving push
remote:
remote: From git://cloud.tsuru.io/blog.git
remote:  * branch            master     -> FETCH_HEAD
remote:
remote:  ---> Installing dependencies
####################################
#          OMIT (see below)        #
####################################
remote:  ---> Restarting your app
remote:
remote:  ---> Deploy done!
remote:
To git@git.tsuru.io:blog.git
   a211fba..bbf5b53  master -> master
```

If you get a "Permission denied (publickey).", make sure you're member of a team and have a public key added to tsuru. To add a key, use key-add command:

```
$ tsuru key-add ~/.ssh/id_rsa.pub
```

You can use `git remote add` to avoid typing the entire remote url every time you want to push:

```
$ git remote add tsuru git@git.tsuru.io:blog.git
```

Then you can run:

```
$ git push tsuru master
Everything up-to-date
```

And you will be also able to omit the `--app` flag from now on:

```
$ tsuru app-info
Application: blog
Repository: git@git.tsuru.io:blog.git
Platform: python
Teams: tsuruteam
Address: blog.cloud.tsuru.io
Units:
+--------------+---------+
| Unit         | State   |
+--------------+---------+
| 9e70748f4f25 | started |
+--------------+---------+
```

For more details on the `--app` flag, see "Guessing app names" section of tsuru command documentation.

5.39.5 Listing dependencies

In the last section we omitted the dependencies step of deploy. In tsuru, an application can have two kinds of dependencies:

- **Operating system dependencies**, represented by packages in the package manager of the underlying operating system (e.g.: `yum` and `apt-get`);
- **Platform dependencies**, represented by packages in the package manager of the platform/language (in Python, `pip`).

All `apt-get` dependencies must be specified in a `requirements.apt` file, located in the root of your application, and pip dependencies must be located in a file called `requirements.txt`, also in the root of the application. Since we will use MySQL with Django, we need to install `mysql-python` package using `pip`, and this package depends on two `apt-get` packages: `python-dev` and `libmysqlclient-dev`, so here is how `requirements.apt` looks like:

```
libmysqlclient-dev
python-dev
```

And here is `requirements.txt`:

```
Django==1.4.1
MySQL-python==1.2.3
South==0.7.6
```

Please notice that we've included `South` too, for database migrations, and `Django`, off-course.

You can see the complete output of installing these dependencies bellow:

```
% git push tsuru master
####################################
#               OMIT               #
####################################
remote: Reading package lists...
remote: Building dependency tree...
remote: Reading state information...
remote: python-dev is already the newest version.
remote: The following extra packages will be installed:
```

```
remote:   libmysqlclient18 mysql-common
remote: The following NEW packages will be installed:
remote:   libmysqlclient-dev libmysqlclient18 mysql-common
remote: 0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
remote: Need to get 2360 kB of archives.
remote: After this operation, 9289 kB of additional disk space will be used.
remote: Get:1 http://archive.ubuntu.com/ubuntu/ quantal/main mysql-common all 5.5.27-0ubuntu2 [13.7 k
remote: Get:2 http://archive.ubuntu.com/ubuntu/ quantal/main libmysqlclient18 amd64 5.5.27-0ubuntu2
remote: Get:3 http://archive.ubuntu.com/ubuntu/ quantal/main libmysqlclient-dev amd64 5.5.27-0ubuntu2
remote: debconf: unable to initialize frontend: Dialog
remote: debconf: (Dialog frontend will not work on a dumb terminal, an emacs shell buffer, or without
remote: debconf: falling back to frontend: Readline
remote: debconf: unable to initialize frontend: Readline
remote: debconf: (This frontend requires a controlling tty.)
remote: debconf: falling back to frontend: Teletype
remote: dpkg-preconfigure: unable to re-open stdin:
remote: Fetched 2360 kB in 1s (1285 kB/s)
remote: Selecting previously unselected package mysql-common.
remote: (Reading database ... 23143 files and directories currently installed.)
remote: Unpacking mysql-common (from .../mysql-common_5.5.27-0ubuntu2_all.deb) ...
remote: Selecting previously unselected package libmysqlclient18:amd64.
remote: Unpacking libmysqlclient18:amd64 (from .../libmysqlclient18_5.5.27-0ubuntu2_amd64.deb) ...
remote: Selecting previously unselected package libmysqlclient-dev.
remote: Unpacking libmysqlclient-dev (from .../libmysqlclient-dev_5.5.27-0ubuntu2_amd64.deb) ...
remote: Setting up mysql-common (5.5.27-0ubuntu2) ...
remote: Setting up libmysqlclient18:amd64 (5.5.27-0ubuntu2) ...
remote: Setting up libmysqlclient-dev (5.5.27-0ubuntu2) ...
remote: Processing triggers for libc-bin ...
remote: ldconfig deferred processing now taking place
remote: sudo: Downloading/unpacking Django==1.4.1 (from -r /home/application/current/requirements.txt
remote:   Running setup.py egg_info for package Django
remote:
remote: Downloading/unpacking MySQL-python==1.2.3 (from -r /home/application/current/requirements.txt
remote:   Running setup.py egg_info for package MySQL-python
remote:
remote:     warning: no files found matching 'MANIFEST'
remote:     warning: no files found matching 'ChangeLog'
remote:     warning: no files found matching 'GPL'
remote: Downloading/unpacking South==0.7.6 (from -r /home/application/current/requirements.txt (line
remote:   Running setup.py egg_info for package South
remote:
remote: Installing collected packages: Django, MySQL-python, South
remote:   Running setup.py install for Django
remote:     changing mode of build/scripts-2.7/django-admin.py from 644 to 755
remote:
remote:     changing mode of /usr/local/bin/django-admin.py to 755
remote:   Running setup.py install for MySQL-python
remote:     building '_mysql' extension
remote:     gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fPIC
remote:     In file included from _mysql.c:36:0:
remote:     /usr/include/mysql/my_config.h:422:0: warning: "HAVE_WCSCOLL" redefined [enabled by defau
remote:     In file included from /usr/include/python2.7/Python.h:8:0,
remote:                      from pymemcompat.h:10,
remote:                      from _mysql.c:29:
remote:     /usr/include/python2.7/pyconfig.h:890:0: note: this is the location of the previous defin
remote:     gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,rel
remote:
remote:     warning: no files found matching 'MANIFEST'
```

```
remote:     warning: no files found matching 'ChangeLog'
remote:     warning: no files found matching 'GPL'
remote:   Running setup.py install for South
remote:
remote: Successfully installed Django MySQL-python South
remote: Cleaning up...
####################################
#               OMIT               #
####################################
To git@git.tsuru.io:blog.git
   a211fba..bbf5b53  master -> master
```

5.39.6 Running the application

As you can see, in the deploy output there is a step described as "Restarting your app". In this step, tsuru will restart your app if it's running, or start it if it's not. But how does tsuru start an application? That's very simple, it uses a Procfile (a concept stolen from Foreman). In this Procfile, you describe how your application should be started. We can use gunicorn, for example, to start our Django application. Here is how the Procfile should look like:

```
web: gunicorn -b 0.0.0.0:$PORT blog.wsgi
```

Now we commit the file and push the changes to tsuru git server, running another deploy:

```
$ git add Procfile
$ git commit -m "Procfile: added file"
$ git push tsuru master
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 326 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote:  ---> tsuru receiving push
remote:
remote:  ---> Installing dependencies
remote: Reading package lists...
remote: Building dependency tree...
remote: Reading state information...
remote: python-dev is already the newest version.
remote: libmysqlclient-dev is already the newest version.
remote: 0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
remote: Requirement already satisfied (use --upgrade to upgrade): Django==1.4.1 in /usr/local/lib/pyt
remote: Requirement already satisfied (use --upgrade to upgrade): MySQL-python==1.2.3 in /usr/local/l
remote: Requirement already satisfied (use --upgrade to upgrade): South==0.7.6 in /usr/local/lib/pyth
remote: Cleaning up...
remote:
remote:  ---> Restarting your app
remote: /var/lib/tsuru/hooks/start: line 13: gunicorn: command not found
remote:
remote:  ---> Deploy done!
remote:
To git@git.tsuru.io:blog.git
   81e884e..530c528  master -> master
```

Now we get an error: `gunicorn: command not found`. It means that we need to add gunicorn to `requirements.txt` file:

```
$ cat >> requirements.txt
gunicorn==0.14.6
^D
```

Now we commit the changes and run another deploy:

```
$ git add requirements.txt
$ git commit -m "requirements.txt: added gunicorn"
$ git push tsuru master
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 325 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote:  ---> tsuru receiving push
remote:
[...]
remote:  ---> Restarting your app
remote:
remote:  ---> Deploy done!
remote:
To git@git.tsuru.io:blog.git
   530c528..542403a  master -> master
```

Now that the app is deployed, you can access it from your browser, getting the IP or host listed in `app-list` and opening it. For example, in the list below:

```
$ tsuru app-list
+-------------+-------------------------+---------------------+--------+
| Application | Units State Summary     | Address             | Ready? |
+-------------+-------------------------+---------------------+--------+
| blog        | 1 of 1 units in-service | blog.cloud.tsuru.io | Yes    |
+-------------+-------------------------+---------------------+--------+
```

We can access the admin of the app in the URL http://blog.cloud.tsuru.io/admin/.

5.39.7 Using services

Now that gunicorn is running, we can accesss the application in the browser, but we get a Django error: *"Can't connect to local MySQL server through socket '/var/run/mysqld/mysqld.sock' (2)"*. This error means that we can't connect to MySQL on localhost. That's because we should not connect to MySQL on localhost, we must use a service. The service workflow can be resumed to two steps:

1. Create a service instance
2. Bind the service instance to the app

But how can I see what services are available? Easy! Use service-list command:

```
$ tsuru service-list
+----------------+-----------+
| Services       | Instances |
+----------------+-----------+
| elastic-search |           |
| mysql          |           |
+----------------+-----------+
```

The output from `service-list` above says that there are two available services: "elastic-search" and "mysql", and no instances. To create our MySQL instance, we should run the service-add command:

```
$ tsuru service-add mysql blogsql
Service successfully added.
```

Now, if we run `service-list` again, we will see our new service instance in the list:

```
$ tsuru service-list
+----------------+-----------+
| Services       | Instances |
+----------------+-----------+
| elastic-search |           |
| mysql          | blogsql   |
+----------------+-----------+
```

To bind the service instance to the application, we use the bind command:

```
$ tsuru bind blogsql
Instance blogsql is now bound to the app blog.

The following environment variables are now available for use in your app:

- MYSQL_PORT
- MYSQL_PASSWORD
- MYSQL_USER
- MYSQL_HOST
- MYSQL_DATABASE_NAME

For more details, please check the documentation for the service, using service-doc command.
```

As you can see from bind output, we use environment variables to connect to the MySQL server. Next step is update `settings.py` to use these variables to connect in the database:

```
import os

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.environ.get('MYSQL_DATABASE_NAME', 'blog'),
        'USER': os.environ.get('MYSQL_USER', 'root'),
        'PASSWORD': os.environ.get('MYSQL_PASSWORD', ''),
        'HOST': os.environ.get('MYSQL_HOST', ''),
        'PORT': os.environ.get('MYSQL_PORT', ''),
    }
}
```

Now let's commit it and run another deploy:

```
$ git add blog/settings.py
$ git commit -m "settings: using environment variables to connect to MySQL"
$ git push tsuru master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 535 bytes, done.
Total 4 (delta 3), reused 0 (delta 0)
remote:
remote:  ---> tsuru receiving push
remote:
```

```
remote:  ---> Installing dependencies
###################################
#               OMIT              #
###################################
remote:
remote:  ---> Restarting your app
remote:
remote:  ---> Deploy done!
remote:
To git@git.tsuru.io:blog.git
   ab4e706..a780de9  master -> master
```

Now if we try to access the admin again, we will get another error: *"Table 'blogsql.django_session' doesn't exist"*. Well, that means that we have access to the database, so bind worked, but we did not set up the database yet. We need to run `syncdb` and `migrate` (if we're using South) in the remote server. We can use run command to execute commands in the machine, so for running `syncdb` we could write:

```
$ tsuru run -- python manage.py syncdb --noinput
Syncing...
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log
Creating table south_migrationhistory
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)

Synced:
 > django.contrib.auth
 > django.contrib.contenttypes
 > django.contrib.sessions
 > django.contrib.sites
 > django.contrib.messages
 > django.contrib.staticfiles
 > django.contrib.admin
 > south

Not synced (use migrations):
 - blog.posts
(use ./manage.py migrate to migrate these)
```

The same applies for `migrate`.

5.39.8 Deployment hooks

It would be boring to manually run `syncdb` and/or `migrate` after every deployment. So we can configure an automatic hook to always run before or after the app restarts.

tsuru parses a file called `app.yaml` and runs restart hooks. As the extension suggests, this is a YAML file, that

contains a list of commands that should run before and after the restart. Here is our example of app.yaml:

```
hooks:
  restart:
    after:
      - python manage.py syncdb --noinput
      - python manage.py migrate
```

For more details, check the *hooks documentation*.

tsuru will look for the file in the root of the project. Let's commit and deploy it:

```
$ git add app.yaml
$ git commit -m "app.yaml: added file"
$ git push tsuru master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 338 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote:  ---> tsuru receiving push
remote:
remote:  ---> Installing dependencies
remote: Reading package lists...
remote: Building dependency tree...
remote: Reading state information...
remote: python-dev is already the newest version.
remote: libmysqlclient-dev is already the newest version.
remote: 0 upgraded, 0 newly installed, 0 to remove and 15 not upgraded.
remote: Requirement already satisfied (use --upgrade to upgrade): Django==1.4.1 in /usr/local/lib/pyt
remote: Requirement already satisfied (use --upgrade to upgrade): MySQL-python==1.2.3 in /usr/local/l
remote: Requirement already satisfied (use --upgrade to upgrade): South==0.7.6 in /usr/local/lib/pyth
remote: Requirement already satisfied (use --upgrade to upgrade): gunicorn==0.14.6 in /usr/local/lib/
remote: Cleaning up...
remote:
remote:  ---> Restarting your app
remote:
remote:  ---> Running restart:after
remote:
remote:  ---> Deploy done!
remote:
To git@git.tsuru.io:blog.git
   a780de9..1b675b8  master -> master
```

It's done! Now we have a Django project deployed on tsuru, using a MySQL service.

5.39.9 Going further

For more information, you can dig into tsuru docs, or read complete instructions of use for the tsuru command.

5.40 Deploying Ruby applications in tsuru

5.40.1 Overview

This document is a hands-on guide to deploying a simple Ruby application in tsuru. The example application will be a very simple Rails project associated to a MySQL service.

5.40.2 Creating the app within tsuru

To create an app, you use app-create command:

```
$ tsuru app-create <app-name> <app-platform>
```

For Ruby, the app platform is, guess what, `ruby`! Let's be over creative and develop a never-developed tutorial-app: a blog, and its name will also be very creative, let's call it "blog":

```
$ tsuru app-create blog ruby
```

To list all available platforms, use platform-list command.

You can see all your applications using app-list command:

```
$ tsuru app-list
+-------------+------------------------+---------+--------+
| Application | Units State Summary    | Address | Ready? |
+-------------+------------------------+---------+--------+
| blog        | 0 of 0 units in-service |         | No     |
+-------------+------------------------+---------+--------+
```

Once your app is ready, you will be able to deploy your code, e.g.:

```
$ tsuru app-list
+-------------+------------------------+-------------+--------+
| Application | Units State Summary    | Address     | Ready? |
+-------------+------------------------+-------------+--------+
| blog        | 0 of 0 units in-service |             | Yes    |
+-------------+------------------------+-------------+--------+
```

5.40.3 Application code

This document will not focus on how to write a blog with Rails, you can clone the entire source direct from GitHub: https://github.com/tsuru/tsuru-ruby-sample. Here is what we did for the project:

1. Create the project (`rails new blog`)
2. Generate the scaffold for Post (`rails generate scaffold Post title:string body:text`)

5.40.4 Git deployment

When you create a new app, tsuru will display the Git remote that you should use. You can always get it using app-info command:

```
$ tsuru app-info --app blog
Application: blog
Repository: git@cloud.tsuru.io:blog.git
```

```
Platform: ruby
Teams: tsuruteam
Address:
```

The git remote will be used to deploy your application using git. You can just push to tsuru remote and your project will be deployed:

```
$  git push git@cloud.tsuru.io:blog.git master
Counting objects: 86, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (75/75), done.
Writing objects: 100% (86/86), 29.75 KiB, done.
Total 86 (delta 2), reused 0 (delta 0)
remote: Cloning into '/home/application/current'...
remote: requirements.apt not found.
remote: Skipping...
remote: /home/application/current /
remote: Fetching gem metadata from https://rubygems.org/.........
remote: Fetching gem metadata from https://rubygems.org/..
####################################
#          OMIT (see below)        #
####################################
remote:  ---> App will be restarted, please check its log for more details...
remote:
To git@cloud.tsuru.io:blog.git
 * [new branch]      master -> master
```

If you get a "Permission denied (publickey).", make sure you're member of a team and have a public key added to tsuru. To add a key, use key-add command:

```
$ tsuru key-add ~/.ssh/id_rsa.pub
```

You can use `git remote add` to avoid typing the entire remote url every time you want to push:

```
$ git remote add tsuru git@cloud.tsuru.io:blog.git
```

Then you can run:

```
$ git push tsuru master
Everything up-to-date
```

And you will be also able to omit the `--app` flag from now on:

```
$ tsuru app-info
Application: blog
Repository: git@cloud.tsuru.io:blog.git
Platform: ruby
Teams: tsuruteam
Address: blog.cloud.tsuru.io
Units:
+--------------+---------+
| Unit         | State   |
+--------------+---------+
| 9e70748f4f25 | started |
+--------------+---------+
```

For more details on the `--app` flag, see "Guessing app names" section of tsuru command documentation.

5.40.5 Listing dependencies

In the last section we omitted the dependencies step of deploy. In tsuru, an application can have two kinds of dependencies:

- **Operating system dependencies**, represented by packages in the package manager of the underlying operating system (e.g.: `yum` and `apt-get`);
- **Platform dependencies**, represented by packages in the package manager of the platform/language (in Ruby, `bundler`).

All `apt-get` dependencies must be specified in a `requirements.apt` file, located in the root of your application, and ruby dependencies must be located in a file called `Gemfile`, also in the root of the application. Since we will use MySQL with Rails, we need to install `mysql` package using `gem`, and this package depends on an `apt-get` package: `libmysqlclient-dev`, so here is how `requirements.apt` looks like:

```
libmysqlclient-dev
```

And here is `Gemfile`:

```
source 'https://rubygems.org'

gem 'rails', '3.2.13'
gem 'mysql'
gem 'sass-rails',   '~> 3.2.3'
gem 'coffee-rails', '~> 3.2.1'
gem 'therubyracer', :platforms => :ruby
gem 'uglifier', '>= 1.0.3'
gem 'jquery-rails'
```

You can see the complete output of installing these dependencies bellow:

```
$ git push tsuru master
#####################################
#                OMIT               #
#####################################
remote: Reading package lists...
remote: Building dependency tree...
remote: Reading state information...
remote: The following extra packages will be installed:
remote:   libmysqlclient18 mysql-common
remote: The following NEW packages will be installed:
remote:   libmysqlclient-dev libmysqlclient18 mysql-common
remote: 0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
remote: Need to get 2360 kB of archives.
remote: After this operation, 9289 kB of additional disk space will be used.
remote: Get:1 http://archive.ubuntu.com/ubuntu/ quantal/main mysql-common all 5.5.27-0ubuntu2 [13.7 k
remote: Get:2 http://archive.ubuntu.com/ubuntu/ quantal/main libmysqlclient18 amd64 5.5.27-0ubuntu2
remote: Get:3 http://archive.ubuntu.com/ubuntu/ quantal/main libmysqlclient-dev amd64 5.5.27-0ubuntu2
remote: Fetched 2360 kB in 2s (1112 kB/s)
remote: Selecting previously unselected package mysql-common.
remote: (Reading database ... 41063 files and directories currently installed.)
remote: Unpacking mysql-common (from .../mysql-common_5.5.27-0ubuntu2_all.deb) ...
remote: Selecting previously unselected package libmysqlclient18:amd64.
remote: Unpacking libmysqlclient18:amd64 (from .../libmysqlclient18_5.5.27-0ubuntu2_amd64.deb) ...
remote: Selecting previously unselected package libmysqlclient-dev.
remote: Unpacking libmysqlclient-dev (from .../libmysqlclient-dev_5.5.27-0ubuntu2_amd64.deb) ...
remote: Setting up mysql-common (5.5.27-0ubuntu2) ...
remote: Setting up libmysqlclient18:amd64 (5.5.27-0ubuntu2) ...
remote: Setting up libmysqlclient-dev (5.5.27-0ubuntu2) ...
```

```
remote: Processing triggers for libc-bin ...
remote: ldconfig deferred processing now taking place
remote: /home/application/current /
remote: Fetching gem metadata from https://rubygems.org/.........
remote: Fetching gem metadata from https://rubygems.org/..
remote: Using rake (10.1.0)
remote: Using i18n (0.6.1)
remote: Using multi_json (1.7.8)
remote: Using activesupport (3.2.13)
remote: Using builder (3.0.4)
remote: Using activemodel (3.2.13)
remote: Using erubis (2.7.0)
remote: Using journey (1.0.4)
remote: Using rack (1.4.5)
remote: Using rack-cache (1.2)
remote: Using rack-test (0.6.2)
remote: Using hike (1.2.3)
remote: Using tilt (1.4.1)
remote: Using sprockets (2.2.2)
remote: Using actionpack (3.2.13)
remote: Using mime-types (1.23)
remote: Using polyglot (0.3.3)
remote: Using treetop (1.4.14)
remote: Using mail (2.5.4)
remote: Using actionmailer (3.2.13)
remote: Using arel (3.0.2)
remote: Using tzinfo (0.3.37)
remote: Using activerecord (3.2.13)
remote: Using activeresource (3.2.13)
remote: Using coffee-script-source (1.6.3)
remote: Using execjs (1.4.0)
remote: Using coffee-script (2.2.0)
remote: Using rack-ssl (1.3.3)
remote: Using json (1.8.0)
remote: Using rdoc (3.12.2)
remote: Using thor (0.18.1)
remote: Using railties (3.2.13)
remote: Using coffee-rails (3.2.2)
remote: Using jquery-rails (3.0.4)
remote: Installing libv8 (3.11.8.17)
remote: Installing mysql (2.9.1)
remote: Using bundler (1.3.5)
remote: Using rails (3.2.13)
remote: Installing ref (1.0.5)
remote: Using sass (3.2.10)
remote: Using sass-rails (3.2.6)
remote: Installing therubyracer (0.11.4)
remote: Installing uglifier (2.1.2)
remote: Your bundle is complete!
remote: Gems in the groups test and development were not installed.
remote: It was installed into ./vendor/bundle
####################################
#               OMIT               #
####################################
To git@cloud.tsuru.io:blog.git
   9515685..d67c3cd  master -> master
```

5.40.6 Running the application

As you can see, in the deploy output there is a step described as "Restarting your app". In this step, tsuru will restart your app if it's running, or start it if it's not. But how does tsuru start an application? That's very simple, it uses a Procfile (a concept stolen from Foreman). In this Procfile, you describe how your application should be started. Here is how the Procfile should look like:

```
web: bundle exec rails server -p $PORT -e production
```

Now we commit the file and push the changes to tsuru git server, running another deploy:

```
$ git add Procfile
$ git commit -m "Procfile: added file"
$ git push tsuru master
####################################
#                OMIT              #
####################################
remote:  ---> App will be restarted, please check its log for more details...
remote:
To git@cloud.tsuru.io:blog.git
   d67c3cd..f2a5d2d  master -> master
```

Now that the app is deployed, you can access it from your browser, getting the IP or host listed in `app-list` and opening it. For example, in the list below:

```
$ tsuru app-list
+-------------+-------------------------+---------------------+--------+
| Application | Units State Summary     | Address             | Ready? |
+-------------+-------------------------+---------------------+--------+
| blog        | 1 of 1 units in-service | blog.cloud.tsuru.io | Yes    |
+-------------+-------------------------+---------------------+--------+
```

5.40.7 Using services

Now that your app is not running with success because the rails can't connect to MySQL. That's because we add a relation between your rails app and a mysql instance. To do it we must use a service. The service workflow can be resumed to two steps:

1. Create a service instance
2. Bind the service instance to the app

But how can I see what services are available? Easy! Use service-list command:

```
$ tsuru service-list
+----------------+-----------+
| Services       | Instances |
+----------------+-----------+
| elastic-search |           |
| mysql          |           |
+----------------+-----------+
```

The output from `service-list` above says that there are two available services: "elastic-search" and "mysql", and no instances. To create our MySQL instance, we should run the service-add command:

```
$ tsuru service-add mysql blogsql
Service successfully added.
```

Now, if we run `service-list` again, we will see our new service instance in the list:

```
$ tsuru service-list
+----------------+-----------+
| Services       | Instances |
+----------------+-----------+
| elastic-search |           |
| mysql          | blogsql   |
+----------------+-----------+
```

To bind the service instance to the application, we use the bind command:

```
$ tsuru bind blogsql
Instance blogsql is now bound to the app blog.

The following environment variables are now available for use in your app:

- MYSQL_PORT
- MYSQL_PASSWORD
- MYSQL_USER
- MYSQL_HOST
- MYSQL_DATABASE_NAME

For more details, please check the documentation for the service, using service-doc command.
```

As you can see from bind output, we use environment variables to connect to the MySQL server. Next step is update `conf/database.yml` to use these variables to connect in the database:

```
production:
  adapter: mysql
  encoding: utf8
  database: <%= ENV["MYSQL_DATABASE_NAME"] %>
  pool: 5
  username: <%= ENV["MYSQL_USER"] %>
  password: <%= ENV["MYSQL_PASSWORD"] %>
  host: <%= ENV["MYSQL_HOST"] %>
```

Now let's commit it and run another deploy:

```
$ git add conf/database.yml
$ git commit -m "database.yml: using environment variables to connect to MySQL"
$ git push tsuru master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 535 bytes, done.
Total 4 (delta 3), reused 0 (delta 0)
remote:
remote:  ---> tsuru receiving push
remote:
remote:  ---> Installing dependencies
####################################
#               OMIT               #
####################################
remote:
remote:  ---> Restarting your app
remote:
remote:  ---> Deploy done!
remote:
To git@cloud.tsuru.io:blog.git
   ab4e706..a780de9  master -> master
```

Now if we try to access the admin again, we will get another error: *"Table 'blogsql.django_session' doesn't exist"*. Well, that means that we have access to the database, so bind worked, but we did not set up the database yet. We need to run `rake db:migrate` in the remote server. We can use run command to execute commands in the machine, so for running `rake db:migrate` we could write:

```
$ tsuru run -- RAILS_ENV=production bundle exec rake db:migrate
==  CreatePosts: migrating ============================================
-- create_table(:posts)
   -> 0.1126s
==  CreatePosts: migrated (0.1128s) =======================================
```

5.40.8 Deployment hooks

It would be boring to manually run `rake db:migrate` after every deployment. So we can configure an automatic hook to always run before or after the app restarts.

tsuru parses a file called `app.yaml` and runs restart hooks. As the extension suggests, this is a YAML file, that contains a list of commands that should run before and after the restart. Here is our example of app.yaml:

```
hooks:
  restart:
    before-each:
      - RAILS_ENV=production bundle exec rake db:migrate
```

For more details, check the *hooks documentation*.

tsuru will look for the file in the root of the project. Let's commit and deploy it:

```
$ git add app.yaml
$ git commit -m "app.yaml: added file"
$ git push tsuru master
####################################
#                OMIT              #
####################################
To git@cloud.tsuru.io:blog.git
   a780de9..1b675b8  master -> master
```

It is necessary to compile de assets before the app restart. To do it we can use the `rake assets:precompile` command. Then let's add the command to compile the assets in app.yaml:

```
hooks:
  restart:
    before:
      - RAILS_ENV=production bundle exec rake assets:precompile
```

```
$ git add app.yaml
$ git commit -m "app.yaml: added file"
$ git push tsuru master
####################################
#                OMIT              #
####################################
To git@cloud.tsuru.io:blog.git
   a780de9..1b675b8  master -> master
```

It's done! Now we have a Rails project deployed on tsuru, using a MySQL service.

Now we can access your *blog app* in the URL http://blog.cloud.tsuru.io/posts/.

5.40.9 Going further

For more information, you can dig into tsuru docs, or read complete instructions of use for the tsuru command.

5.41 tsuru-admin usage

tsuru-admin command supports administrative operations on a tsuru server. It can be compiled with:

```
$ go get github.com/tsuru/tsuru/cmd/tsuru-admin
```

To use *tsuru-admin* commands you should be an admin user. To be an admin user you should be in an admin team.

5.41.1 Setting a target

The target for the tsuru-admin command should point to the *listen* address configured in your tsuru.conf file.

```
listen: ":8080"
```

```
$ tsuru-admin target-add default tsuru.myhost.com:8080
$ tsuru-admin target-set default
```

5.41.2 Commands

All the "container*"" commands below only exist when using the docker provisioner.

containers-move

```
$ tsuru-admin containers-move <from host> <to host>
```

It allows you to move all containers from one host to another. This is useful when doing maintenance on hosts. <from host> and <to host> must be host names of existing docker servers. They can either be added to the docker:servers entry in the tsuru.conf file or added dynamically if using other schedulers, see docker schedulers for more details.

This command will go through the following steps:

- Enumerate all units at the origin host;
- For each unit, create a new unit at the destination host;
- Erase each unit from the origin host.

container-move

```
$ tsuru-admin container-move <container id> <to host>
```

This command allow you to specify a container id and a destination host, this will create a new container on the destination host and remove the container from its previous host.

containers-rebalance

```
$ tsuru-admin containers-rebalance [--dry]
```

Instead of specifying hosts as in the containers-move command, this command will automatically choose to which host each unit should be moved, trying to distribute the units as evenly as possible.

The –dry flag runs the balancing algorithm without doing any real modification. It will only print which units would be moved and where they would be created.

All the "platform*"" commands below only exist when using the docker provisioner.

platform-add

```
$ tsuru-admin platform-add <name> [--dockerfile]
```

This command allow you to add a new platform to your tsuru installation. It will automatically create and build a whole new platform on tsuru server and will allow your users to create apps based on that platform.

The –dockerfile flag is an URL to a dockerfile which will create your platform.

platform-update

```
$ tsuru-admin platform-update <name> [--dockerfile]
```

This command allow you to update a platform in your tsuru installation. It will automatically rebuild your platform and will flag apps to update platform on next deploy.

The –dockerfile flag is an URL to a dockerfile which will update your platform.

5.42 Docker Provisioner Architecture

This document describes how tsuru works when configured with docker provisioner. Docker

The docker provisioner is responsible for provisioning your application units. Everytime your perform an action in your application tsuru repasses the request with specific parameters to the configured provisioner. In this document you will learn how the docker provisioner reacts facing those actions.

Given the app creation -> deploy workflow.

5.42.1 App Provisioning

When you create an application tsuru asks the provisioner to provision the application, the docker provisioner will do nothing in this action, the only change is that tsuru creates the application on the database. Docker provisioner will wait until you perform a deploy, so it can create a base image to your application.

5.42.2 Deployment

When you perform a git push into your application repository on tsuru the custom pre-receive git hook is triggered, this hook will ask tsuru to deploy your application, tsuru will then repass the action to docker. Docker will run a container, clone your application code to it and install all dependencies specified by your application, then it will generate an image of that container and store its id on the database, this container is then destroyed and a new one is run starting your application. This allows an easy and fast scalability for your application, whenever you need a new unit tsuru can deploy one in a few seconds.

Every deploy will trigger this process, resulting in a new image with the deployed version and new dependencies if any.

5.42.3 HTTP Routing

Because containers are ephemeral their routes changes everytime a deploy is performed. So we need an easy and fast way to manage routes to containers, by default the docker provisioner uses Hipache router. Routes to containers are managed transparently by the docker provisioner. The hipache router also acts as a load balancer to the containers, distributing traffic using a round robin algorithm.

5.43 Schedulers

tsuru uses schedulers to chooses which node an unit should be deployed. There are two schedulers: *round robin* and *segregate scheduler*.

5.43.1 Segregate scheduler

Segregate scheduler is a scheduler that segregates the units between nodes by team.

First, what you need to do is to define a relation between a pool, teams and nodes. And then, the scheduler deploys the app unit on the pool where a node is related to its team.

- Pool1 - team1, team2 - node1
- Pool2 - team2 - node3, node4
- Pool3 (fallback) - <no teams> - node2

Configuration and setup

To use the *segregate scheduler* you shoud enable the segregate mode in *tsuru.conf* and make sure that the details about the scheduler storage (redis) is also configured:

```
docker:
  segregate: true
  scheduler:
    redis-server: 127.0.0.1:6379
    redis-prefix: docker-cluster
```

Adding a pool

Using *tsuru-admin* you create a pool:

```
$ tsuru-admin docker-pool-add pool1
```

Removing a pool

A pool is removable if it don't have any node associated with it. To remove a pool you do:

```
$ tsuru-admin docker-pool-remove pool1
```

Listing a pool

To list pools you do:

```
$ tsuru-admin docker-pool-list
+-------+------------------+-----------+
| Pools | Nodes            | Teams     |
+-------+------------------+-----------+
| pool1 | node1, node2     | team1     |
| pool2 | node3            | team2     |
+-------+------------------+-----------+
```

Adding node to a pool

You can use the *tsuru-admin* to add nodes:

```
$ tsuru-admin docker-node-add pool1 http://localhost:4243
```

Removing a node

You can use the *tsuru-admin* to remove nodes:

```
$ tsuru-admin docker-node-remove pool1 http://localhost:4243
Node successfully removed.
```

List nodes

```
$ tsuru-admin docker-nodes-list
+-----------+
| Address   |
+-----------+
| node1     |
| node2     |
+-----------+
```

Adding teams to a pool

You can add one or more teams at once.

```
$ tsuru-admin docker-pool-teams-add pool1 team1
```

```
$ tsuru-admin docker-pool-teams-add pool1 team1 team2 team3
```

Removing teams from a pool

You can remove one or more teams at once.

```
$ tsuru-admin docker-pool-teams-remove pool1 team1
```

```
$ tsuru-admin docker-pool-teams-remove pool1 team1 team2 team3
```

5.44 tsr 0.3.0 release notes

Welcome to tsr 0.3.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsuru 0.2.x or older versions.

5.44.1 What's new in tsr 0.3.0

Support Docker 0.7.x and other improvements

- Fixed the 42 layers problem.
- Support all Docker storages.
- Pull image on creation if it does not exists.
- BUGFIX: when using segregatedScheduler, the provisioner fails to get the proper host address.
- BUGFIX: units losing access to services on deploy bug.

Improvements related to Services

- *bind* is atomic.
- *service-add* is atomic
- Service instance name is unique.
- Add support to bind an app without units.

Collector ticker time is configurable

Now you can define the collector ticker time. To do it just set on tsuru.conf:

```
collector:
    ticker-time: 120
```

The default value is 60 seconds.

Other improvements and bugfixes

- *unit-remove* does not block util all units are removed.
- BUGFIX: send on closed channel: https://github.com/tsuru/tsuru/issues/624.
- Api handler that returns information about all deploys.
- Refactored quota backend.
- New lisp platform. Thanks to Nick Ricketts.

5.44.2 Backwards incompatible changes

tsuru 0.3.0 handles quota in a brand new way. Users upgrading from 0.2.x need to run a migration script in the database. There are two scripts available: one for installations with quota enabled and other for installations without quota.

The easiest script is recommended for environments where quota is disabled, you'll need to run just a couple of commands in MongoDB:

```
% mongo tsuru
MongoDB shell version: x.x.x
connecting to: tsuru
> db.users.update({}, {$set: {quota: {limit: -1}}});
> db.apps.update({}, {$set: {quota: {limit: -1}}});
```

In environments where quota is enabled, the script is longer, but still simple:

```
db.quota.find().forEach(function(quota) {
    if(quota.owner.indexOf("@") > -1) {
        db.users.update({email: quota.owner}, {$set: {quota: {limit: quota.limit, inuse: quota.items
    } else {
        db.apps.update({name: quota.owner}, {$set: {quota: {limit: quota.limit, inuse: quota.items.le
    }
});
```

db.apps.update({quota: null}, {$set: {quota: {limit: -1}}}); db.users.update({quota: null}, {$set: {quota: {limit: -1}}}); db.quota.remove()

The best way to run it is saving it to a file and invoke MongoDB with the file parameter:

```
% mongo tsuru <filename.js>
```

5.45 tsr 0.3.1 release notes

Welcome to tsr 0.3.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsuru 0.3.0 or older versions.

5.45.1 What's new in tsr 0.3.1

5.45.2 Backwards incompatible changes

5.46 tsr 0.3.10 release notes

Welcome to tsr 0.3.10!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.9 or older versions.

5.46.1 What's new in tsr 0.3.10

API

- Improve feedback for duplicated users (issue #693)

Docker provisioner

- Update docker-cluster library, to fix the behavior of the default scheduler (issue #716)
- Improve debug logs for SSH (issue #665)
- Fix URL for listing containers by app

5.46.2 Backwards incompatible changes

tsr 0.3.10 did not introduce any incompatible changes.

5.47 tsr 0.3.11 release notes

Welcome to tsr 0.3.11!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.10 or older versions.

5.47.1 What's new in tsr 0.3.11

API

- Added app team owner - #619
- Expose public url in *create-app* - #724

Docker provisioner

- Add support to custom memory - #434

5.47.2 Backwards incompatible changes

All existing apps have no team owner. You can run the mongodb script below to automatically set the first existing team in the app as team owner.

```
db.apps.find({ teamowner: { $exists: false }}).forEach(
    function(app) {
        app.teamowner = app.teams[0];
        db.apps.save(app);
    }
);
```

5.48 tsr 0.3.12 release notes

Welcome to tsr 0.3.12!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.11 or older versions.

5.48.1 What's new in tsr 0.3.12

Docker provisioner

- integrated the segregated scheduler with owner team - #753

5.48.2 Backwards incompatible changes

tsr 0.3.12 did not introduce any incompatible changes.

5.49 tsr 0.3.2 release notes

Welcome to tsr 0.3.2!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.1 or older versions.

5.49.1 What's new in tsr 0.3.2

Segregated scheduler

- Support more than one team per scheduler
- Fix the behavior of the segregated scheduler
- Improve documentation of the scheduler

API

- Improve administrative API registration

Other improvements and bugfixes

- Do not run restart on unit-add (nor unit-remove)
- Improve node management in the Docker provisioner
- Rebuild app image on every 10 deployment

5.49.2 Backwards incompatible changes

tsr 0.3.2 does not introduce any incompatible changes.

5.50 tsr 0.3.3 release notes

Welcome to tsr 0.3.3!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.2 or older versions.

5.50.1 What's new in tsr 0.3.3

Queue

- Add an option to use Redis instead of beanstalk for work queue

In order to use Redis, you need to change the configuration file:

```
queue: redis
redis-queue:
  host: "localhost"
  port: 6379
  db: 4
  password: "your-password"
```

All settings are optional (`queue` will still default to "beanstalkd"), refer to *configuration docs* for more details.

Other improvements and bugfixes

- Do not depend on Docker code
- Improve the layout of the documentation
- Fix multiple data races in tests
- [BUGFIX] fix bug with unit-add and application image
- [BUGFIX] fix image replication on docker nodes

5.50.2 Backwards incompatible changes

tsr 0.3.3 does not introduce any incompatible changes.

5.51 tsr 0.3.4 release notes

Welcome to tsr 0.3.4!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.3 or older versions.

5.51.1 What's new in tsr 0.3.4

Documentation improvements

- Improvements in the layout of the documentation

Bugfixes

- Swap address and cname on apps when running swap
- Always pull the image before creating the container

5.51.2 Backwards incompatible changes

tsr 0.3.4 does not introduce any incompatible changes.

5.52 tsr 0.3.5 release notes

Welcome to tsr 0.3.5!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.4 or older versions.

5.52.1 What's new in tsr 0.3.5

Bugfixes

- Fix administrative API for Docker provisioner

5.52.2 Backwards incompatible changes

tsr 0.3.5 does not introduce any incompatible changes.

5.53 tsr 0.3.6 release notes

Welcome to tsr 0.3.6!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.5 or older versions.

5.53.1 What's new in tsr 0.3.6

Application state control

- Add new functionality to the API and provisoners: stop and starting an App

Services

- Add support for plans in services

5.53.2 Backwards incompatible changes

tsr 0.3.6 does not introduce any incompatible changes.

5.54 tsr 0.3.7 release notes

Welcome to tsr 0.3.7!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.6 or older versions.

5.54.1 What's new in tsr 0.3.7

API

- Improve administrative API for the Docker provisioner
- Store deploy metadata
- Improve healthcheck (ping MongoDB before marking the API is ok)
- Expose owner of the app in the app-info API

5.54.2 Backwards incompatible changes

tsr 0.3.7 does not introduce any incompatible changes.

5.55 tsr 0.3.8 release notes

Welcome to tsr 0.3.8!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.8 or older versions.

5.55.1 What's new in tsr 0.3.8

API

- Expose deploys of the app in the app-info API

Docker

- deploy hook support enviroment variables with space.

5.55.2 Backwards incompatible changes

tsr 0.3.7 does not introduce any incompatible changes.

5.56 tsr 0.3.9 release notes

Welcome to tsr 0.3.9!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.8 or older versions.

5.56.1 What's new in tsr 0.3.9

API

- Login expose *is_admin* info.
- Changed get environs output data.

5.56.2 Backwards incompatible changes

tsr 0.3.9 has changed the api output data for get environs from an app.

You should use *tsuru* cli 0.8.10 version.

5.57 tsr 0.4.0 release notes

Welcome to tsr 0.4.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.3.x or older versions.

5.57.1 What's new in tsr 0.4.0

- redis queue backend was refactored.
- fixed output when service doesn't export environment variables (#772)

Docker

- refactored unit creation to be more atomic
- support for unit-agent (#633) - tsuru unit agent repository: https://github.com/tsuru/tsuru-unit-agent
- added an administrative command to move and rebalance containers between nodes (#646) - docs about rebalance: http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#containers-rebalance
- memory swap limit is configurable (#764)
- added a command to add a new platform (#780) - docs about *platform-add* command: http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#platform-add

5.57.2 Backwards incompatible changes

The s3 integration on app creation was removed. The config properties *bucket-support*, *aws:iam aws:s3* was removed too.

You should use *tsuru* cli 0.9.0 and *tsuru-admin* 0.3.0 version.

5.58 tsr 0.5.0 release notes

Welcome to tsr 0.5.0!

These release notes cover the new features and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.4.0 or older versions.

5.58.1 What's new in tsr 0.5.0

Stability and Consistency

One of the main feature on this release is improve the stability and consitency of the tsuru API.

- prevent inconsitency caused by problems on deploy (#803) / (#804)
- units information is not updated by collector (#806)
- fixed log listener on multiple API hosts (#762)
- prevent inconsitency caused by simultaneous operations in an application (#789)
- prevent inconsitency cause by simultaneous `env-set` calls (#820)
- store information about errors and identify flawed application deployments (#816)

Buildpack

tsuru now supports deploying applications using Heroku Buildpacks.

Buildpacks are useful if you're interested in following Heroku's best practices for building applications or if you are deploying an application that already runs on Heroku.

tsuru uses Buildstep Docker image to deploy applications using buildpacks. For more information, take a look at the buildpacks documentation page: http://docs.tsuru.io/en/latest/using/buildpacks.html.

Other features

- filter application logs by unit (#375)
- support for deployments with archives, which enables the use of the `pre-receive` Git hook, and also deployments without Git (#458, #442 and #701)
- stop and start commands (#606)
- oauth support (#752)
- platform update command (#780)
- support services with *https* endpoint (#812) / (#821)

- grouping nodes by pool in segregate scheduler. For more information you can see the docs about the segregate scheduler: *Schedulers*.

Platforms

- *deployment hooks* support for static and PHP applications (#607)
- new platform: buildpack (used for buildpack support)

5.58.2 Backwards incompatible changes

- Juju provisioner was removed. This provisioner was not being maintained. A possible idea is to use Juju in the future to provision the tsuru nodes instead of units
- ELB router was removed. This router was used only by juju.
- `tsr admin` was removed.
- The field `units` was removed from the collection `apps`. Information about units are now available in the provisioner. Now the unit state is controlled by provisioner. If you are upgrading tsuru from 0.4.0 or an older version you should run the MongoDB script bellow, where the *docker* collection name is the name configured by *docker:collection* in *tsuru.conf*:

```
var migration = function(doc) {
    doc.units.forEach(function(unit){
        db.docker.update({"id": unit.name}, {$set: {"status": unit.state}});
    });
};

db.apps.find().forEach(migration);
```

- The scheduler collection has changed to group nodes by pool. If you are using this scheduler you shoul run the MongoDB script bellow:

```
function idGenerator(id) {
    return id.replace(/\d+/g, "")
}

var migration = function(doc) {
    var id = idGenerator(doc._id);
    db.temp_scheduler_collection.update(
        {teams: doc.teams},
        {$push: {nodes: doc.address},
         $set: {teams: doc.teams, _id: id}},
        {upsert: true});
}
db.docker_scheduler.find().forEach(migration);
db.temp_scheduler_collection.renameCollection("docker_scheduler", true);
```

You can implement your own *idGenerator* to return the name for the new pools. In our case the *idGenerator* generates an id based on node name. It makes sense because we use the node name to identify a node group.

5.58.3 Features deprecated in 0.5.0

Beanstalkd queue backend will be removed in 0.6.0.

5.59 tsr 0.5.1 release notes

Welcome to tsr 0.5.1!

These release notes cover the new features, bug fixes and backwards incompatible changes you'll want to be aware of when upgrading from tsr 0.5.0 or older versions.

5.59.1 What's new in tsr 0.5.1

- **tsr api** now checks tsuru.conf file and refuse to start if it is misconfigured. It's also possible to exclusively test the config file with the -t flag. i.e.: running "tsr api -t". (#714).
- new command in the `tsuru-admin`: the command fix-containers will look for broken containers and fix their configuration within the router, and in the database

5.59.2 Bug fixes

- Do not lock application on `tsuru run`

5.59.3 Backwards incompatible changes

- **tsr collector** is no more. In the 0.5.0 release, collector got much less responsibilities, and now it does nothing, because it no longer exists. The last of its responsibilities is now available in the `tsuru-admin fix-containers` command.

5.60 tsuru-admin 0.3.0 release notes

Welcome to tsuru-admin 0.3.0!

These release notes cover the new features when upgrading from tsuru-admin 0.2.x or older versions.

5.60.1 What's new in tsr 0.3.0

New commands:

- containers-move (#756): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#containers-move
- container-move (#754): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#container-move
- containers-rebalance (#646): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#containers-rebalance
- platform-add (#283): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#platform-add

5.61 tsuru-admin 0.4.0 release notes

Welcome to tsuru-admin 0.4.0!

These release notes cover the new features when upgrading from tsuru-admin 0.3.x or older versions.

5.61.1 What's new in tsr 0.4.0

New commands:

- platform-update(#780): http://docs.tsuru.io/en/latest/apps/tsuru-admin/usage.html#platform-update

5.62 tsuru-admin 0.4.1 release notes

Welcome to tsuru-admin 0.4.1!

tsuru-admin 0.4.1 includes a bug fix in the authentication error detection when communicating with the Tsuru API.

5.63 tsuru 0.10.0 release notes

Welcome to tsuru 0.10.0!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.9.x or older versions.

5.63.1 What's new in tsuru 0.10.0

- added stop command.

5.64 tsuru 0.10.1 release notes

Welcome to tsuru 0.10.1.

tsuru 0.10.1 includes a bug fix in the authentication error detection when communicating with the Tsuru API.

5.65 tsuru 0.8.10 release notes

Welcome to tsuru 0.8.10!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.10 or older versions.

5.65.1 What's new in tsuru 0.8.10

Support *tsr* 0.3.9.

5.66 tsuru 0.8.11 release notes

Welcome to tsuru 0.8.11!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.11 or older versions.

5.66.1 What's new in tsuru 0.8.11

- new plugin system - #737

  Now is possible customize tsuru client installing and creating plugins. See the docs for more info

- app team owner is configurable - #620

  Now you can define the app team owner on *app-create*:

```
tsuru app-create appname platform -t teamname
```

5.67 tsuru 0.8.6 release notes

Welcome to tsuru 0.8.6!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.5 or older versions.

5.67.1 What's new in tsuru 0.8.6

Improvements related to Services

- Added confirmation on *service-remove* command.

5.68 tsuru 0.8.7 release notes

Welcome to tsuru 0.8.7!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.6 or older versions.

5.68.1 What's new in tsuru 0.8.7

Added support for service plans

You can use the *service-info* command to the see the plans of a service:

```
$ tsuru service-info redis
Info for "redis"

Plans
+-------+-----------------------------------------------------------------------------------------------
| Name  | Description
+-------+-----------------------------------------------------------------------------------------------
| basic | Is a dedicated instance. With 1GB of memory.
| plus  | Is 3 dedicated instances. With 1GB of memory and HA and failover support via redis-sentinel
+-------+-----------------------------------------------------------------------------------------------
```

And on *service-add* the plan should be defined:

```
$ tsuru service-add redis myredis basic
Service successfully added.
```

Improvements on app-info and app-list

Now the app address and the app cname is displayed on *app-info* and *app-list* command.

5.69 tsuru 0.8.8 release notes

Welcome to tsuru 0.8.8!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.7 or older versions.

5.69.1 What's new in tsuru 0.8.8

Bugfix

- Fixed a bug on *service-info* command.

5.70 tsuru 0.8.9 release notes

Welcome to tsuru 0.8.9!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.8 or older versions.

5.70.1 What's new in tsuru 0.8.9

Improvements on app-info

Now the app owner is displayed on *app-info* command

5.71 tsuru 0.8.9.1 release notes

Welcome to tsuru 0.8.9.1!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.9 or older versions.

5.71.1 What's new in tsuru 0.8.9.1

Improvements on app-info

Now the number of deploys is displayed on *app-info* command

5.72 tsuru 0.9.0 release notes

Welcome to tsuru 0.9.0!

These release notes cover the new features you'll want to be aware of when upgrading from tsuru 0.8.x or older versions.

5.72.1 What's new in tsuru 0.9.0

- fixed app-list output.

5.73 Guide to create tsuru cli plugins

5.73.1 Installing a plugin

Let's install a plugin. There are two ways to install. The first way is to move your plugin to *$HOME/.tsuru/plugins*. The other way is to use *tsuru plugin-install* command.

tsuru plugin-install will download your plugin file to *$HOME/.tsuru/plugins*. The *tsuru plugin-install* sintaxe is:

```
$ tsuru plugin-install <plugin-name> <plugin-url>
```

5.73.2 Listing installed plugins

To list all installed plugins you can use the *tsuru* <*plugin-list*> command:

```
$ tsuru plugin-list
plugin1
plugin2
```

5.73.3 Executing a plugin

To execute a plugin just follow this pattern *tsuru* <*plugin-name*> <*args*>:

```
$ tsuru <plugin-name>
<plugin-output>
```

5.73.4 Removing a plugin

To remove a plugin just use the *plugin-remove* command passing the <*plugin-name*> as argument:

```
$ tsuru plugin-remove <plugin-name>
Plugin "<plugin-name>" successfully removed!
```

5.73.5 Creating your own plugin

Everything you need to do is to create a new *executable*. You can use bash, python, ruby, eg.

Let's create a *Hello world* plugin that prints *hello world* as output. Let's use *bash* to write our new plugin.

```
#! /bin/bash
echo "hello world!"
```

You can use the gist (gist.github.com) as host for your plugin.

To install your plugin you can use *tsuru plugin-install* command.