tsuru-client

Release 0.15.2

May 17, 2017

Contents

tsuru is the command line utility used by application developers, that will allow users to create, list, bind and manage apps.

Note: This documentation is a reference of **tsuru** command line interface. If you want know about how to use tsuru, you should see the tsuru documentation.

There are several ways to install *tsuru-client*:

- Downloading binaries (Mac OS X and Linux)
- Using homebrew (Mac OS X only)
- Using the PPA (Ubuntu only)
- Using AUR (ArchLinux only)
- Build from source (Linux and Mac OS X)

CHAPTER 1

Downloading binaries (Mac OS X and Linux)

We provide pre-built binaries for OS X and Linux, only for the amd64 architecture. You can download these binaries directly from the releases page of the project:

- tsuru: https://github.com/tsuru/tsuru-client/releases

CHAPTER 2

Using homebrew (Mac OS X only)

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

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

Now you can install tsuru:

```
$ brew install tsuru
```

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

```
$ brew update
$ brew upgrade tsuru
```

For more details on taps, check homebrew documentation.

NOTE: tsuru requires Go 1.2 or higher. Make sure you have the last version of Go installed in your system.

CHAPTER 3

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-client
```

CHAPTER 4

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
```

CHAPTER 5

Build from source (Linux and Mac OS X)

Note: If you're feeling adventurous, you can try it on other systems, like FreeBSD, OpenBSD or even Windows. Please let us know about your progress!

tsuru client 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-client/tsuru
```

CHAPTER 6

Reference

Managing remote tsuru server endpoints

In tsuru, a target is the address of the remote tsuru server.

Each target is identified by a label and a HTTP/HTTPS address. The client requires at least one target to connect to, there's no default target. A user may have multiple targets, but he/she will be able to use only per session.

Add a new target

```
$ tsuru target-add <label> <target> [--set-current|-s]
```

Adds a new entry to the list of available targets

Flags:

-s, --set-current (= false) Add and define the target as the current target

Minimum # of arguments: 2

List existing targets

```
$ tsuru target-list
```

Displays the list of targets, marking the current.

Other commands related to target:

- target-add: adds a new target to the list of targets
- target-set: defines one of the targets in the list as the current target
- target-remove: removes one target from the list

Set a target as current

```
$ tsuru target-set <label>
```

Change current target (tsuru server)

Minimum # of arguments: 1

Removes an existing target

```
$ tsuru target-remove
```

Remove a target from target-list (tsuru server)

Minimum # of arguments: 1

Check current version

```
$ tsuru version
```

display the current version

Authentication

Create a user

```
$ tsuru user-create <email>
```

Creates a user within tsuru remote server. It will ask for the password before issue the request.

Minimum # of arguments: 1

Remove your user from tsuru server

```
$ tsuru user-remove [email]
```

Remove currently authenticated user from remote tsuru server. Since there cannot exist any orphan teams, tsuru will refuse to remove a user that is the last member of some team. If this is your case, make sure you remove the team using *team-remove* before removing the user.

Maximum # of arguments: 1

Retrieve information about the current user

```
$ tsuru user-info
```

Displays information about the current user.

Login

```
$ tsuru login [email]
```

Initiates a new tsuru session for a user. If using tsuru native authentication scheme, it will ask for the email and the password and check if the user is successfully authenticated. If using OAuth, it will open a web browser for the user to complete the login.

After that, the token generated by the tsuru server will be stored in `${HOME}/.tsuru_token`.

All tsuru actions require the user to be authenticated (except `tsuru login` and `tsuru version`).

Logout

```
$ tsuru logout
```

Logout will terminate the session with the tsuru server.

Change user's password

```
$ tsuru change-password
```

Changes the password of the logged in user. It will ask for the current password, the new and the confirmation.

Resets user's password

```
$ tsuru reset-password <email> [--token|-t <token>]
```

Resets the user password.

This process is composed of two steps:

1. Generate a new token
2. Reset the password using the token

In order to generate the token, users should run this command without the –token flag. The token will be mailed to the user.

With the token in hand, the user can finally reset the password using the –token flag. The new password will also be mailed to the user.

Flags:

-t, –token (= "") Token to reset the password

Minimum # of arguments: 1

Show current valid API token

```
$ tsuru token-show
```

Shows API token for the user. This token allow authenticated API calls to tsuru and will never expire. This is useful for integrating CI servers with tsuru.

The key will be generated the first time this command is called. See `tsuru token-regenerate` if you need to invalidate an existing token.

Regenerate API token

```
$ tsuru token-regenerate
```

Generates a new API token. This invalidates previously generated API tokens.

Team management

Create a new team

```
$ tsuru team-create <teamname>
```

Create a team for the user. tsuru requires a user to be a member of at least one team in order to create an app or a service instance.

When you create a team, you're automatically member of this team.

Minimum # of arguments: 1

Remove a team from tsuru

```
$ tsuru team-remove <team-name>
```

Removes a team from tsuru server. You're able to remove teams that you're member of. A team that has access to any app cannot be removed. Before removing a team, make sure it does not have access to any app (see "app-grant" and "app-revoke" commands for details).

Flags:

-y, --assume-yes (= false) Don't ask for confirmation.

Minimum # of arguments: 1

List teams current user is member

```
$ tsuru team-list
```

List all teams that you are member.

Add a user to a team

```
$ tsuru team-user-add <teamname> <useremail>
```

Adds a user to a team. You need to be a member of the team to be able to add another user to it.

Minimum # of arguments: 2

Remove a user from a team

```
$ tsuru team-user-remove <teamname> <useremail>
```

Removes a user from a team. You need to be a member of the team to be able to remove a user from it.

Minimum # of arguments: 2

List members of a team

```
$ tsuru team-user-list <teamname>
```

List members of a team.

Minimum # of arguments: 1

Applications

Guessing application names

Some application related commands that are described below have the optional parameter `-a/--app`, used to specify the name of the application.

If this parameter is omitted, tsuru will try to *guess* the application's name based on the git repository's configuration. It will try to find a remote labeled **tsuru**, and parse its URL.

If no remote named **tsuru** is found, tsuru will try to use the current directory name as the application's name.

List of available platforms

```
$ tsuru platform-list
```

Lists the available platforms. All platforms displayed in this list may be used to create new apps (see app-create).

List of available plans

```
$ tsuru plan-list [--human]
```

List available plans that can be used when creating an app.

Flags:

-h, --human (= false) Humanized units for memory and swap.

Create an application

```
$ tsuru app-create <appname> <platform> [--plan/-p plan_name] [--team/-t (team owner)] [-o/--pool poo
```

Creates a new app using the given name and platform. For tsuru, a platform is provisioner dependent. To check the available platforms, use the command `tsuru platform-list` and to add a platform use the command `tsuru-admin platform-add`.

In order to create an app, you need to be member of at least one team. All teams that you are member (see `tsuru team-list`) will be able to access the app.

The `--platform` parameter is the name of the platform to be used when creating the app. This will define how tsuru understands and executes your app. The list of available platforms can be found running `tsuru platform-list`.

The `--plan` parameter defines the plan to be used. The plan specifies how computational resources are allocated to your application. Typically this means limits for memory and swap usage, and how much cpu share is allocated. The list of available plans can be found running `tsuru plan-list`.

If this parameter is not informed, tsuru will choose the plan with the `default` flag set to true.

The `--team` parameter describes which team is responsible for the created app, this is only needed if the current user belongs to more than one team, in which case this parameter will be mandatory.

The `--pool` parameter defines which pool your app will be deployed. This is only needed if you have more than one pool associated with your teams.

Flags:

-o, –pool (= “”) Pool to deploy your app

-p, –plan (= “”) The plan used to create the app

-t, –team (= “”) Team owner app

Minimum # of arguments: 2

Change the application plan

```
$ tsuru app-plan-change <plan_name> [-a/--app appname] [-y/--assume-yes]
```

Change the plan of the application.

Flags:

-a, –app (= “”) The name of the app.

Definition list ends without a blank line; unexpected unindent.

-y, --assume-yes (= false) Don’t ask for confirmation.

Minimum # of arguments: 1

Remove an application

```
$ tsuru app-remove [-a/--app appname] [-y/--assume-yes]
```

Removes an application. If the app is bound to any service instance, all binds will be removed before the app gets deleted (see `tsuru service-unbind`).

You need to be a member of a team that has access to the app to be able to remove it (you are able to remove any app that you see in `tsuru app-list`).

Flags:

-a, –app (= “”) The name of the app.

Definition list ends without a blank line; unexpected unindent.

-y, --assume-yes (= false) Don’t ask for confirmation.

List your applications

```
$ tsuru app-list
```

Lists all apps that you have access to. App access is controlled by teams. If your team has access to an app, then you have access to it.

Flags can be used to filter the list of applications.

Flags:

-l, --locked (= false) Display only applications that are locked

Option list ends without a blank line; unexpected unindent.

-n, –name (= “”) Filter applications by name

-p, –platform (= “”) Display only applications that use the given platform

-t, –team (= “”) Display only applications owned by the given team

-u, –user (= “”) Display only applications owner by the given user

Display information about an application

```
$ tsuru app-info [-a/--app appname]
```

Shows information about a specific app. Its state, platform, git repository, etc. You need to be a member of a team that has access to the app to be able to see information about it.

Flags:

-a, –app (= “”) The name of the app.

Show logs of an application

```
$ tsuru app-log [-a/--app appname] [-l/--lines numberOfLines] [-s/--source source] [-u/--unit unit]
```

Shows log entries for an application. These logs include everything the application send to stdout and stderr, alongside with logs from tsuru server (deployments, restarts, etc.)

The `--lines` flag is optional and by default its value is 10.

The `--source` flag is optional and allows filtering logs by log source (e.g. application, tsuru api).

The `--unit` flag is optional and allows filtering by unit. It’s useful if your application has multiple units and you want logs from a single one.

The `--follow` flag is optional and makes the command wait for additional log output

Flags:

-a, –app (= “”) The name of the app.

Definition list ends without a blank line; unexpected unindent.

-f, --follow (= false) Follow logs

-l, --lines (= 10) The number of log lines to display

Option list ends without a blank line; unexpected unindent.

-s, –source (= “”) The log from the given source

-u, –unit (= “”) The log from the given unit

Stop an application

```
$ tsuru app-stop [-a/--app appname] [-p/--process processname]
```

Stops an application, or one of the processes of the application.

Flags:

-a, –app (= “”) The name of the app.

-p, –process (= “”) Process name

Start an application

```
$ tsuru app-start [-a/--app appname] [-p/--process processname]
```

Starts an application, or one of the processes of the application.

Flags:

-a, –app (= “”) The name of the app.

-p, –process (= “”) Process name

Restart an application

```
$ tsuru app-restart [-a/--app appname] [-p/--process processname]
```

Restarts an application, or one of the processes of the application.

Flags:

-a, –app (= “”) The name of the app.

-p, –process (= “”) Process name

Swap the routing between two applications

```
$ tsuru app-swap <app1-name> <app2-name> [-f/--force]
```

Swaps routing between two apps. This allows zero downtime and makes rollback as simple as swapping the applications back.

Use `--force` if you want to swap applications with a different number of units or different platform without confirmation.

Flags:

-f, --force	(= false) Force Swap among apps with different number of units or different platform.

Minimum # of arguments: 2

Add new units to an application

```
$ tsuru unit-add <# of units> [-a/--app appname] [-p/--process processname]
```

Adds new units to a process of an application. You need to have access to the app to be able to add new units to it.

Flags:

-a, –app (= “”) The name of the app.

-p, –process (= “”) Process name

Minimum # of arguments: 1

Remove units from an application

```
$ tsuru unit-remove <# of units> [-a/--app appname] [-p/-process processname]
```

Removes units from a process of an application. You need to have access to the app to be able to remove units from it.

Flags:

-a, –app (= “”) The name of the app.

-p, –process (= “”) Process name

Minimum # of arguments: 1

Change an application team owner

```
$ tsuru app-set-team-owner <new-team-owner> [-a/--app appname]
```

Sets owner team for an application.

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1

Allow a team to access an application

```
$ tsuru app-grant <teamname> [-a/--app appname]
```

Allows a team to access an application. You need to be a member of a team that has access to the app to allow another team to access it. grants access to an app to a team.

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1

Revoke a team’s access to an application

```
$ tsuru app-revoke <teamname> [-a/--app appname]
```

Revokes the permission to access an application from a team. You need to have access to the application to revoke access from a team.

An application cannot be orphaned, so it will always have at least one authorized team.

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1

Run an arbitrary command in application's containers

```
$ tsuru app-run <command> [commandarg1] [commandarg2] ... [commandargn] [-a/--app appname] [-o/--once
```

Runs an arbitrary command in application's containers. The base directory for all commands is the root of the application.

If you use the `--once` flag tsuru will run the command only in one unit. Otherwise, it will run the command in all units.

Flags:

-a, –app (= "") The name of the app.

Definition list ends without a blank line; unexpected unindent.

-o, --once (= false) Running only one unit

Minimum # of arguments: 1

Open a shell to an application's container

```
$ tsuru app-shell [unit-id] -a/--app <appname>
```

Opens a remote shell inside unit, using the API server as a proxy. You can access an app unit just giving app name, or specifying the id of the unit. You can get the ID of the unit using the app-info command.

Flags:

-a, –app (= "") The name of the app.

Deploy

```
$ tsuru app-deploy [-a/--app <appname>] <file-or-dir-1> [file-or-dir-2] ... [file-or-dir-n]
```

Deploys set of files and/or directories to tsuru server. Some examples of calls are:

```
$ tsuru app-deploy .
$ tsuru app-deploy myfile.jar Procfile
$ tsuru app-deploy mysite
```

Flags:

-a, –app (= "") The name of the app.

Minimum # of arguments: 1

List deploys

```
$ tsuru app-deploy-list [-a/--app <appname>]
```

List information about deploys for an application.

Flags:

-a, –app (= "") The name of the app.

Rollback deploy

```
$ tsuru app-deploy-rollback [-a/--app appname] [-y/--assume-yes] <image-name>
```

Deploys an existing image for an app. You can list available images with *tsuru app-deploy-list*.

Flags:

-a, –app (= “”) The name of the app.

Definition list ends without a blank line; unexpected unindent.

-y, --assume-yes (= false) Don’t ask for confirmation.

Minimum # of arguments: 1

Public Keys

Add SSH public key

```
$ tsuru key-add <key-name> <path/to/key/file.pub> [-f/--force]
```

Sends your public key to the git server used by tsuru.

Flags:

-f, --force (= false) Force overriding the key if it already exists

Minimum # of arguments: 2

Remove SSH public key

```
$ tsuru key-remove <key-name> [-y/--assume-yes]
```

Removes your public key from the git server used by tsuru. The key will be removed from the current logged in user.

Flags:

-y, --assume-yes (= false) Don’t ask for confirmation.

Minimum # of arguments: 1

List SSH public keys

```
$ tsuru key-list [-n/--no-truncate]
```

Lists the public keys registered in the current user account.

Flags:

-n, --no-truncate (= false) disable truncation of key content

Services

List available services and instances

```
$ tsuru service-list
```

Retrieves and shows a list of services the user has access. If there are instances created for any service they will also be shown.

Create a new service instance

```
$ tsuru service-add <servicename> <serviceinstancename> [plan] [-t/--team-owner <team>]
```

Creates a service instance of a service. There can later be binded to applications with `tsuru service-bind`.

This example shows how to add a new instance of **mongodb** service, named **tsuru_mongodb** with the plan **small**:

```
$ tsuru service-add mongodb tsuru_mongodb small -t myteam
```

Flags:

-t, –team-owner (= “”) the team that owns the service (mandatory if the user is member of more than one team)

Minimum # of arguments: 2 Maximum # of arguments: 3

Remove a service instance

```
$ tsuru service-remove <serviceinstancename> [--assume-yes]
```

Destroys a service instance. It can’t remove a service instance that is bound to an app, so before remove a service instance, make sure there is no apps bound to it (see `tsuru service-info` command).

Flags:

-y, --assume-yes (= false) Don’t ask for confirmation, just remove the service.

Minimum # of arguments: 1

Display information about a service

```
$ tsuru service-info <service-name>
```

Displays a list of all instances of a given service (that the user has access to), and apps bound to these instances.

Minimum # of arguments: 1

Check if a service instance is up

```
$ tsuru service-status <service-instance-name>
```

Displays the status of the given service instance. For now, it checks only if the instance is “up” (receiving connections) or “down” (refusing connections).

Minimum # of arguments: 1

Check if a service instance is up

```
$ tsuru service-doc <service-name>
```

Shows the documentation of a service.

Minimum # of arguments: 1

Bind an application to a service instance

```
$ tsuru service-bind <service-instance-name> [-a/--app appname]
```

Binds an application to a previously created service instance. See `tsuru service-add` for more details on how to create a service instance.

When binding an application to a service instance, tsuru will add new environment variables to the application. All environment variables exported by bind will be private (not accessible via `tsuru env-get`).

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1

Unbind an application from a service instance

```
$ tsuru service-unbind <instancename> [-a/--app appname]
```

Unbinds an application from a service instance. After unbinding, the instance will not be available anymore. For example, when unbinding an application from a MySQL service, the application would lose access to the database.

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1

Grant access to a team in service instance

```
$ tsuru service-instance-grant <service_instance_name> <team_name>
```

Grant access to team in a service instance.

Minimum # of arguments: 2

Revoke access to a team in service instance

```
$ tsuru service-instance-revoke <service_instance_name> <team_name>
```

Revoke access to team in a service instance.

Minimum # of arguments: 2

Environment variables

Applications running on tsuru should use environment variables to handle configurations. As an example, if you need to connect with a third party service like twitter's API, your application is going to need things like an `api_key`.

In tsuru, the recommended way to expose these values to applications is using environment variables. To make this easy, tsuru provides commands to set and get environment variables in a running application.

Set environment variables

```
$ tsuru env-set <NAME=value> [NAME=value] ... [-a/--app appname] [-p/--private]
```

Sets environment variables for an application.

Flags:

-a, –app (= "") The name of the app.

Definition list ends without a blank line; unexpected unindent.

-p, --private (= false) Private environment variables

Minimum # of arguments: 1

Show environment variables

```
$ tsuru env-get [-a/--app appname] [ENVIRONMENT_VARIABLE1] [ENVIRONMENT_VARIABLE2] ...
```

Retrieves environment variables for an application.

Flags:

-a, –app (= "") The name of the app.

Unset environment variables

```
$ tsuru env-unset <ENVIRONMENT_VARIABLE1> [ENVIRONMENT_VARIABLE2] ... [ENVIRONMENT_VARIABLEN] [-a/--a
```

Unset environment variables for an application.

Flags:

-a, –app (= "") The name of the app.

Minimum # of arguments: 1

Plugin management

Plugins allow extending tsuru client's functionality. Plugins are executables existing in `$HOME/.tsuru/plugins`.

Installing a plugin

There are two ways to install. The first way is to manually copy your plugin to `$HOME/.tsuru/plugins`. The other way is to use `tsuru plugin-install` command.

Install a plugin

```
$ tsuru plugin-install <plugin-name> <plugin-url>
```

Downloads the plugin file. It will be copied to `$HOME/.tsuru/plugins`.

Minimum # of arguments: 2

List installed plugins

```
$ tsuru plugin-list
```

List installed tsuru plugins.

Remove a plugin

```
$ tsuru plugin-remove <plugin-name>
```

Removes a previously installed tsuru plugin.

Minimum # of arguments: 1

Executing a plugin

To execute a plugin just follow the pattern `tsuru <plugin-name> <args>`:

```
$ tsuru <plugin-name>
<plugin-output>
```

CNAME management

Add a CNAME to the app

```
$ tsuru cname-add <cname> [<cname> ...] [-a/--app appname]
```

Adds a new CNAME to the application.

It will not manage any DNS register, it's up to the user to create the DNS register. Once the app contains a custom CNAME, it will be displayed by "app- list" and "app-info".

Flags:

-a, –app (= "") The name of the app.

Minimum # of arguments: 1

Remove a CNAME from the app

```
$ tsuru cname-remove <cname> [<cname> ...] [-a/--app appname]
```

Removes a CNAME from the application. This undoes the change that cname-add does.

After unsetting the CNAME from the app, `tsuru app-list` and `tsuru app- info` will display the internal, unfriendly address that tsuru uses.

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1

Pool

List available pool

```
$ tsuru pool-list
```

List all pools available for deploy.

Change an app’s pool

```
$ tsuru app-pool-change <pool_name> [-a/--app appname]
```

Change app pool. You need to have access to the pool to be able to do it.

Flags:

-a, –app (= “”) The name of the app.

Minimum # of arguments: 1