tsuru-client

Release 1.0.1

July 24, 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

List users

```
$ tsuru user-list [--user/-u useremail] [--role/-r role [-c/--context-value value]]
```

List all users in tsuru. It may also filter users by user email or role name with context value.

Flags:

-c, --context-value	(= “”) Filter user by role context value
-r, --role	(= “”) Filter user by role
-u, --user	(= “”) Filter user by user email

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 [--user/-u useremail]
```

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.

Flags:

-u, --user (= "") Shows API token for the given user email

Regenerate API token

```
$ tsuru token-regenerate [--user/-u useremail]
```

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

Flags:

-u, --user (= "") Generates a new API token for the given user email

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.

Authorization

List all available permissions

```
$ tsuru permission-list [-t/--tree]
```

Lists all permissions available to use when defining roles.

Flags:

-t, --tree (= false) Show permissions in tree format.

Create a new role

```
$ tsuru role-add <role-name> <context-type> [--description/-d description]
```

Create a new role for the specified context type. Valid context types are:

- global
- app
- team
- pool
- iaas
- service
- service-instance

The `--description` parameter sets a description for your role. It is an optional parameter, and if its not set the role will only not have a description associated.

Flags:

-d, --description (= "") Role description

Minimum # of arguments: 2

Remove a role

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

Remove an existing role.

Flags:

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

Minimum # of arguments: 1

List all created roles

```
$ tsuru role-list
```

List all existing roles.

Info about specific role

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

Get information about specific role.

Minimum # of arguments: 1

Add a permission to a role

```
$ tsuru role-permission-add <role-name> <permission-name>...
```

Add a new permission to an existing role.

Minimum # of arguments: 2

Remove a permission from a role

```
$ tsuru role-permission-remove <role-name> <permission-name>
```

Remove a permission from an existing role.

Minimum # of arguments: 2

Assign a role to a user

```
$ tsuru role-assign <role-name> <user-email> [<context-value>]
```

Assign an existing role to a user with some context value.

Minimum # of arguments: 2

Dissociate a role from a user

```
$ tsuru role-dissociate <role-name> <user-email> [<context-value>]
```

Dissociate an existing role from a user for some context value.

Minimum # of arguments: 2

List default roles

```
$ tsuru role-default-list
```

List all roles set as default on any event.

Add new default roles

```
$ tsuru role-default-add [--user-create <role name>]... [--team-create <role name>]...
```

Add a new default role on a specific event.

Flags:

--team-create	(= []) role added to user when a new team is created
--user-create	(= []) role added to user when user is created

Remove default roles

```
$ tsuru role-default-remove [--user-create <role name>]... [--team-create <role name>]...
```

Remove a default role from a specific event.

Flags:

--team-create	(= []) role added to user when a new team is created
--user-create	(= []) role added to user when user is created

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.

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 [--bytes]
```

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

Flags:

-b, --bytes	(= false) bytesized units for memory and swap.

Create an application

```
$ tsuru app-create <appname> <platform> [--plan/-p plan name] [--router/-r router name] [--team/-t te
```

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 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 `--router` parameter defines the router to be used. The list of available routers can be found running `tsuru router-list`.

If this parameter is not informed, tsuru will choose the router 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.

The `--description` parameter sets a description for your app. It is an optional parameter, and if its not set the app will only not have a description associated.

The `--tag` parameter sets a tag to your app. You can set multiple `--tag` parameters.

The `--router-opts` parameter allow passing custom parameters to the router used by the application's plan. The key and values used depends on the router implementation.

Flags:

-d, --description	(= "") App description
-g, --tag	(= []) App tag
-o, --pool	(= "") Pool to deploy your app
-p, --plan	(= "") The plan used to create the app
-r, --router	(= "") The router used by the app
--router-opts	(= {}) Router options
-t, --team	(= "") Team owner app

Minimum # of arguments: 2

Update an application

```
$ tsuru app-update [-a/--app appname] [--description/-d description] [--plan/-p plan name] [--router/
```

Updates an app, changing its description, tags, plan or pool information.

The `--description` parameter sets a description for your app.

The `--plan` parameter changes the plan of your app.

The `--router` parameter changes the router of your app.

The `--pool` parameter changes the pool of your app.

The `--team-owner` parameter sets owner team for an application.

The `--tag` parameter sets a tag for your app. You can set multiple `--tag` parameters.

Flags:

-a, --app	(= “”) The name of the app.
-d, --description	(= “”) App description
-g, --tag	(= []) App tag
-o, --pool	(= “”) App pool
-p, --plan	(= “”) App plan
-r, --router	(= “”) App router
-t, --team-owner	(= “”) App team owner

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.
-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:

-g, --tag	(= []) Filter applications by tag. Can be used multiple times
-l, --locked	(= false) Filter applications by lock status
-n, --name	(= “”) Filter applications by name
-o, --pool	(= “”) Filter applications by pool
-p, --platform	(= “”) Filter applications by platform
-q	(= false) Display only applications name

-s, --status	(= “”) Filter applications by unit status. Accepts multiple values separated by commas. Possible values can be: building, created, starting, error, started, stopped, asleep
-t, --team	(= “”) Filter applications by team owner
-u, --user	(= “”) Filter applications by owner

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

The `--no-date` flag is optional and makes the log output without date.

The `--no-source` flag is optional and makes the log output without source information, useful to very dense logs.

Flags:

-a, --app	(= “”) The name of the app.
-f, --follow	(= false) Follow logs
-l, --lines	(= 10) The number of log lines to display
--no-date	(= false) No date information
--no-source	(= false) No source information
-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] [-c/--cname-only]
```

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.

Use `--cname-only` if you want to swap all cnames except the default cname of application

Flags:

-c, --cname-only (= false) Swap all cnames except the default cname.

-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

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.

-i, --isolated (= false) Running in ephemeral container

-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>] [-i/--image <image_url>] <file-or-dir-1> [file-or-dir-2] ...
```

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
$ tsuru app-deploy -i http://registry.mysite.com:5000/image-name
```

Flags:

-a, --app (= "") The name of the app.

-i, --image (= "") The image to deploy in app

-m, --message (= "") A message describing this deploy

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.

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

Minimum # of arguments: 1 Maximum # of arguments: 1

Set application certificate

```
$ tsuru certificate-set [-a/--app appname] [-c/--cname CNAME] [certificate] [key]
```

Creates or update a TLS certificate into the specific app.

Flags:

-a, --app	(= “”) The name of the app.
-c, --cname	(= “”) App CNAME

Minimum # of arguments: 2

Unset application certificate

```
$ tsuru certificate-unset [-a/--app appname] [-c/--cname CNAME]
```

Unset a TLS certificate from a specific app.

Flags:

-a, --app	(= “”) The name of the app.
-c, --cname	(= “”) App CNAME

List application certificates

```
$ tsuru certificate-list [-a/--app appname] [-r/--raw]
```

List App TLS certificates.

Flags:

-a, --app	(= “”) The name of the app.
-r, --raw	(= false) Display raw certificates

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.

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

Create a service instance

```
$ tsuru service-instance-add <service-name> <service-instance-name> [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-instance-add mongodb tsuru_mongodb small -t myteam
```

Flags:

-d, --description (= "") service instance description

-g, --tag (= []) service instance tag

-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

Update a service instance

```
$ tsuru service-instance-update <service-name> <service-instance-name> [-d/--description description]
```

Updates a service instance of a service.

The –description parameter sets a description for your service instance.

The –tag parameter adds a tag to your service instance. This parameter may be used multiple times.

Flags:

-d, --description (= "") service instance description

-g, --tag (= []) service instance tag

Minimum # of arguments: 2

Remove a service instance

```
$ tsuru service-instance-remove <service-name> <service-instance-name> [--assume-yes] [--unbind]
```

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-instance-info` command).

Flags:

-u, --unbind (= false) Don't ask for confirmation, just remove all applications bound.

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

Minimum # of arguments: 2

Display the status of a service instance

```
$ tsuru service-instance-status <service-name> <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: 2

Display the information of a service instance

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

Displays the information of the given service instance.

Minimum # of arguments: 2

Bind an application to a service instance

```
$ tsuru service-instance-bind <service-name> <service-instance-name> [-a/--app appname] [--no-restart
```

Binds an application to a previously created service instance. See `tsuru service-instance-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.
--no-restart	(= false) Binds an application to a service instance without restart the application

Minimum # of arguments: 2

Unbind an application from a service instance

```
$ tsuru service-instance-unbind <service-name> <service-instance-name> [-a/--app appname] [--no-resta
```

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.
--no-restart	(= false) Unbinds an application from a service instance without restart the application

Minimum # of arguments: 2

Grant access to a team in service instance

```
$ tsuru service-instance-grant <service-name> <service-instance-name> <team-name>
```

Grant access to team in a service instance.

Minimum # of arguments: 3

Revoke access to a team in service instance

```
$ tsuru service-instance-revoke <service-name> <service-instance-name> <team-name>
```

Revoke access to team in a service instance.

Minimum # of arguments: 3

Service Management

These commands manage entire services and not particular instances.

Create a service

```
$ tsuru service-create path/to/manifest [- for stdin]
```

Creates a service based on a passed manifest. The manifest format should be a yaml and follow the standard described in the documentation (should link to it here)

Minimum # of arguments: 1

Destroy a service

```
$ tsuru service-destroy <servicename>
```

removes a service from catalog

Flags:

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

Minimum # of arguments: 1 Maximum # of arguments: 1

Update a service

```
$ tsuru service-update <path/to/manifest>
```

Update service data, extracting it from the given manifest file.

Minimum # of arguments: 1

Generate a manifest template file

```
$ tsuru service-template
```

e.g.: $ tsuru service-template template

Generates a manifest template file and places it in current directory

Add documentation to a service

```
$ tsuru service-doc-add <service> <path/to/docfile>
```

Update service documentation, extracting it from the given file.

Minimum # of arguments: 2

Get documentation of a service

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

Shows service documentation.

Minimum # of arguments: 1

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] [--no-restart]
```

Sets environment variables for an application.

Flags:

-a, --app	(= "") The name of the app.
--no-restart	(= false) Sets environment varibles without restart the application
-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.
--no-restart	(= false) Unset environment variables without restart the application

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.

Events

List all events

```
$ tsuru event-list [-k kindName]
```

Lists events possibly filtering them.

Flags:

-k, --kind	(= “”) Filter events by kind name
-o, --owner	(= “”) Filter events by owner name
-r, --running	(= false) Shows only currently running events
-t, --target	(= “”) Filter events by target type
-v, --target-value	(= “”) Filter events by target value

Show detailed information about an event

```
$ tsuru event-info <event-id>
```

Show detailed information about one single event.

Minimum # of arguments: 1 Maximum # of arguments: 1

Cancel an event

```
$ tsuru event-cancel <event-id> <reason> [-y]
```

Cancel a running event.

Flags:

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

Minimum # of arguments: 2

Container management

All the **container** commands below only exist when using the docker provisioner.

Moves single container

```
$ tsuru container-move <container id> <to host>
```

Move specified container to another 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.

Minimum # of arguments: 2

Moves all containers from on node

```
$ tsuru containers-move <from host> <to host>
```

Move all containers from one host to another. This command 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 nodes.

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.

Minimum # of arguments: 2

Node management

Add a new node

```
$ tsuru node-add [param_name=param_value]... [--register]
```

Creates or registers a new node in the cluster. By default, this command will call the configured IaaS to create a new machine. Every param will be sent to the IaaS implementation.

IaaS providers should have been previously configured in the `tsuru.conf` file. See tsuru.conf reference docs for more information.

If using an IaaS to create a node is not wanted it's possible to simply register an existing node with the `--register` flag.

Parameters with special meaning:

iaas=<iaas name> Which iaas provider should be used, if not set tsuru will use the default iaas specified in tsuru.conf file.

template=<template name> A machine template with predefined parameters, additional parameters will override template ones. See 'machine-template-add' command.

address=<api url> Only used if `--register` flag is used. Should point to the endpoint of a working server.

pool=<pool name> Mandatory parameter specifying to which pool the added node will belong. Available pools can be listed with the `pool-list` command.

Flags:

--cacert	(= "") Path to CA file tsuru should trust.
--clientcert	(= "") Path to client TLS certificate file.
--clientkey	(= "") Path to client TLS key file.
--register	(= false) Registers an existing docker endpoint, the IaaS won't be called.

Minimum # of arguments: 1

List nodes in cluster

```
$ tsuru node-list [--filter/-f <metadata>=<value>]...
```

Lists nodes in the cluster. It will also show you metadata associated to each node and the IaaS ID if the node was added using tsuru IaaS providers.

Using the `-f/--filter` flag, the user is able to filter the nodes that appear in the list based on the key pairs displayed in the metadata column. Users can also combine filters using `-f` multiple times.

Flags:

-f, --filter	(= {}) Filter by metadata name and value
-q	(= false) Display only nodes IP address

Update a node

```
$ tsuru node-update <address> [param_name=param_value...] [--disable] [--enable]
```

Modifies metadata associated to a node. If a parameter is set to an empty value, it will be removed from the node's metadata.

If the `--disable` flag is used, the node will be marked as disabled and the scheduler won't consider it when selecting a node to receive containers.

Flags:

--disable	(= false) Disable node in scheduler.
--enable	(= false) Enable node in scheduler.

Minimum # of arguments: 1

Remove a node

```
$ tsuru node-remove <address> [--no-rebalance] [--destroy] [-y]
```

Removes a node from the cluster.

By default tsuru will redistribute all containers present on the removed node among other nodes. This behavior can be inhibited using the `--no-rebalance` flag.

If the node being removed was created using a IaaS provider tsuru will NOT destroy the machine on the IaaS, unless the `--destroy` flag is used.

Flags:

--destroy	(= false) Destroy node from IaaS
--no-rebalance	(= false) Do not rebalance containers from removed node.
-y, --assume-yes	(= false) Don't ask for confirmation.

Minimum # of arguments: 1

Rebalance containers in nodes

```
$ tsuru node-rebalance [--dry] [-y/--assume-yes] [-m/--metadata <metadata>=<value>]... [-a/--app <app
```

Move units among nodes trying to create a more even distribution. This command will automatically choose to which node 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.

Flags:

-a, --app	(= []) Filter by app name
--dry	(= false) Dry run, only shows what would be done
-m, --metadata	(= {}) Filter by host metadata
-y, --assume-yes	(= false) Don't ask for confirmation.

Node Containers management

Add a new node container

```
$ tsuru node-container-add <name> [-o/--pool poolname] [-r/--raw path=value]... [docker run flags]...
```

Add new node container or overwrite existing one. If the pool name is omitted the node container will be valid for all pools.

Flags:

-e, --env	(= []) Set environment variables
--image	(= "") Image that will be used
--log-driver	(= "") Logging driver for container
--log-opt	(= {}) Log driver options

--net	(= “”) Connect a container to a network
-o, --pool	(= “”) Pool to add container config. If empty it’ll be a default entry to all pools.
-p, --publish	(= []) Publish a container’s port(s) to the host
--privileged	(= false) Give extended privileges to this container
-r, --raw	(= {}) Add raw parameter to node container api call
--restart	(= “”) Restart policy to apply when a container exits
-v, --volume	(= []) Bind mount a volume

Minimum # of arguments: 1 Maximum # of arguments: 1

Delete an existing node container

```
$ tsuru node-container-delete <name> [-p/--pool poolname] [-k/--kill] [-y]
```

Delete existing node container.

Flags:

-k, --kill	(= false) Kill running containers.
-p, --pool	(= “”) Pool to remove container config. If empty the default node container will be removed.
-y, --assume-yes	(= false) Don’t ask for confirmation.

Minimum # of arguments: 1 Maximum # of arguments: 1

Update an existing node container

```
$ tsuru node-container-update <name> [-o/--pool poolname] [-r/--raw path=value]... [docker run flags]
```

Update an existing node container. If the pool name is omitted the default configuration will be updated. When updating node containers the specified configuration will be merged with the existing configuration.

Flags:

-e, --env	(= []) Set environment variables
--image	(= “”) Image that will be used
--log-driver	(= “”) Logging driver for container
--log-opt	(= {}) Log driver options
--net	(= “”) Connect a container to a network
-o, --pool	(= “”) Pool to update container config. If empty it’ll be a default entry to all pools.
-p, --publish	(= []) Publish a container’s port(s) to the host
--privileged	(= false) Give extended privileges to this container
-r, --raw	(= {}) Add raw parameter to node container api call
--restart	(= “”) Restart policy to apply when a container exits

-v, --volume (= []) Bind mount a volume

Minimum # of arguments: 1 Maximum # of arguments: 1

List existing node containers

```
$ tsuru node-container-list
```

List all existing node containers.

Flags:

-q (= false) Show only names of existing node containers.

Show information abort a node container

```
$ tsuru node-container-info <name>
```

Show details about a single node container.

Minimum # of arguments: 1 Maximum # of arguments: 1

Upgrade node container version on docker nodes

```
$ tsuru node-container-upgrade <name> [-p/--pool poolname] [-y]
```

Upgrade version and restart node containers.

Flags:

-p, --pool (= "") Pool to upgrade container. If empty it'll be a default entry to all pools.

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

Minimum # of arguments: 1 Maximum # of arguments: 1

Machine management

List IaaS machines

```
$ tsuru machine-list
```

Lists all machines created using an IaaS provider. These machines were created with the `docker-node-add` command.

Destroy IaaS machine

```
$ tsuru machine-destroy <machine id>
```

Destroys an existing machine created using a IaaS.

Minimum # of arguments: 1

List machine templates

```
$ tsuru machine-template-list
```

Lists all machine templates.

Add machine template

```
$ tsuru machine-template-add <name> <iaas> <param>=<value>...
```

Creates a new machine template.

Templates can be used with the `docker-node-add` command running it with the `template=<template name>` parameter. Templates can contain a list of parameters that will be sent to the IaaS provider.

Minimum # of arguments: 3

Remove machine template

```
$ tsuru machine-template-remove <name>
```

Removes an existing machine template.

Minimum # of arguments: 1

Update machine template

```
$ tsuru machine-template-update <name> <param>=<value>... [-i/--iaas <iaas_name>]
```

Update an existing machine template.

Flags:

-i, --iaas	(= "") The iaas name to be used

Minimum # of arguments: 2

Pool management

Add a new pool

```
$ tsuru pool-add <pool> [-p/--public] [-d/--default] [--provisioner <name>] [-f/--force]
```

Adds a new pool.

Each docker node added using `node-add` command belongs to one pool. Also, when creating a new application a pool must be chosen and this means that all units of the created application will be spawned in nodes belonging to the chosen pool.

Flags:

-d, --default	(= false) Make pool default (when none is specified during `app-create` this pool will be used)
-f, --force	(= false) Force overwrite default pool

-p, --public (= false) Make pool public (all teams can use it)

--provisioner (= "") Provisioner associated to the pool (empty for default docker provisioner)

Minimum # of arguments: 1

Update pool attributes

```
$ tsuru pool-update <pool> [--public=true/false] [--default=true/false] [-f/--force]
```

Updates attributes for a pool.

Flags:

--default (= not set) Make pool default (when none is specified during `app-create` this pool will be used)

-f, --force (= false) Force pool to be default.

--public (= not set) Make pool public (all teams can use it)

Minimum # of arguments: 1

Remove a pool

```
$ tsuru pool-remove <pool> [-y]
```

Remove an existing pool.

Flags:

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

Minimum # of arguments: 1

Healer

List latest healing events

```
$ tsuru healing-list [--node] [--container]
```

List healing history for nodes or containers.

Flags:

--container (= false) List only healing process started for containers

--node (= false) List only healing process started for nodes

Show node healing config information

```
$ tsuru node-healing-info
```

Show the current configuration for active healing nodes.

Update node healing configuration

```
$ tsuru node-healing-update [-p/--pool pool] [--enable] [--disable] [--max-unresponsive <seconds>] [-
```

Update node healing configuration

Flags:

--disable	(= false) Disable active node healing
--enable	(= false) Enable active node healing
--max-unresponsive	(= -1) Number of seconds tsuru will wait for the node to notify it's alive
--max-unsuccessful	(= -1) Number of seconds tsuru will wait for the node to run successful checks
-p, --pool	(= "") The pool name to which the configuration will apply. If unset it'll be set as default for all pools.

Delete node healing configuration

```
$ tsuru node-healing-delete [-p/--pool pool] [--enabled] [--max-unresponsive] [--max-unsuccessful]
```

Delete a node healing configuration entry.

If `--pool` is provided the configuration entries from the specified pool will be removed and the default value will be used.

If `--pool` is not provided the configuration entry will be removed from the default configuration.

Flags:

--enabled	(= false) Remove the 'enabled' configuration option
--max-unresponsive	(= false) Remove the 'max-unresponsive' configuration option
--max-unsuccessful	(= false) Remove the 'max-unsuccessful' configuration option
-p, --pool	(= "") The pool name from where the configuration will be removed. If unset it'll delete the default healing configuration.
-y, --assume-yes	(= false) Don't ask for confirmation.

Platform management

> **Warning:** All the **platform** commands below only exist when using the docker provisioner.

Add a new platform

```
$ tsuru platform-add <platform name> [--dockerfile/-d Dockerfile] [--image/-i image]
```

Adds a new platform to tsuru.

The name of the image can be automatically inferred in case you're using an official platform. Check https://github.com/tsuru/platforms for a list of official platforms and instructions on how to create a custom platform.

Examples:

```
tsuru platform-add java # uses official tsuru/java image from docker
hub         tsuru platform-add java -i registry.company.com/tsuru/java
# uses custom Java image                    tsuru platform-add java -d
/data/projects/java/Dockerfile # uses local Dockerfile          tsuru
platform-add java -d https://platforms.com/java/Dockerfile # uses
remote Dockerfile
```

Flags:

-d, --dockerfile	(= "") URL or path to the Dockerfile used for building the image of the platform
-i, --image	(= "") Name of the prebuilt Docker image

Minimum # of arguments: 1

Update an existing platform

```
$ tsuru platform-update <platform name> [--dockerfile/-d Dockerfile] [--disable/--enable] [--image/-i
```

Updates a platform in tsuru.

The name of the image can be automatically inferred in case you're using an official platform. Check https://github.com/tsuru/platforms for a list of official platforms.

The flags –enable and –disable can be used for enabling or disabling a platform.

Examples:

```
tsuru platform-update java # uses official tsuru/java image from docker hub
tsuru platform-update java -i registry.company.com/tsuru/java # uses custom
Java image tsuru platform-update java -d /data/projects/java/Dockerfile # uses
local Dockerfiletsuru platform-update java -d https://platforms.com/java/Dockerfile
# uses remote Dockerfile
```

Flags:

-d, --dockerfile	(= "") URL or path to the Dockerfile used for building the image of the platform
--disable	(= false) Disable the platform
--enable	(= false) Enable the platform
-i, --image	(= "") Name of the prebuilt Docker image

Minimum # of arguments: 1

Remove an existing platform

```
$ tsuru platform-remove <platform name> [-y]
```

Remove a platform from tsuru. This command will fail if there are application still using the platform.

Flags:

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

Minimum # of arguments: 1

Plan management

Create a new plan

```
$ tsuru plan-create <name> -c cpushare [-m memory] [-s swap] [--default]
```

Creates a new plan for being used when creating apps.

Flags:

-c, --cpushare	(= 0) Relative cpu share each unit will have available. This value is unitless and relative, so specifying the same value for all plans means all units will equally share processing power.
-d, --default	(= false) Set plan as default, this will remove the default flag from any other plan. The default plan will be used when creating an application without explicitly setting a plan.
-m, --memory	(= "0") Amount of available memory for units in bytes or an integer value followed by M, K or G for megabytes, kilobytes or gigabytes respectively.
-s, --swap	(= "0") Amount of available swap space for units in bytes or an integer value followed by M, K or G for megabytes, kilobytes or gigabytes respectively.

Minimum # of arguments: 1

Remove an existing plan

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

Removes an existing plan. It will no longer be available for newly created apps. However, this won't change anything for existing apps that were created using the removed plan. They will keep using the same value amount of resources described by the plan.

Minimum # of arguments: 1

List available routers

```
$ tsuru router-list
```

List all routers available for app creation.

Auto Scale

List auto scale events

```
$ tsuru node-autoscale-list [--page/-p 1]
```

List node auto scale history.

Flags:

-p, --page (= 1) Current page

Run auto scale process algorithm once

```
$ tsuru node-autoscale-run [-y/--assume-yes]
```

Run node auto scale checks once. This command will work even if `docker:auto- scale:enabled` config entry is set to false. Auto scaling checks may trigger the addition, removal or rebalancing of nodes, as long as these nodes were created using an IaaS provider registered in tsuru.

Flags:

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

Show auto scale rules

```
$ tsuru node-autoscale-info
```

Display the current configuration for tsuru autoscale, including the set of rules and the current metadata filter.

The metadata filter is the value that defines which node metadata will be used to group autoscale rules. A common approach is to use the "pool" as the filter. Then autoscale can be configured for each matching rule value.

Set a new auto scale rule

```
$ tsuru node-autoscale-rule-set [-f/--filter-value <pool name>] [-c/--max-container-count 0] [-m/--ma
```

Creates or update an auto-scale rule. Using resources limitation (amount of container or memory usage).

Flags:

-c, --max-container-count (= 0) The maximum amount of containers on every node. Might be zero, which means no maximum value. Whenever this value is reached, tsuru will trigger a new auto scale event.

-d, --scale-down-ratio (= 1.33) The ratio for triggering an scale down event. The default value is 1.33, which mean that whenever it gets one third of the resource utilization (memory ratio or container count).

--disable (= false) A boolean flag indicating whether the rule should be disabled

--enable (= false) A boolean flag indicating whether the rule should be enabled

-f, --filter-value (= "") The pool name matching the rule. This is the unique identifier of the rule.

-m, --max-memory-ratio (= 0) The maximum memory usage per node. 0 means no limit, 1 means 100%. It is fine to use values greater than 1, which means that tsuru will overcommit memory in nodes. Keep in mind that container count has higher precedence than memory ratio, so if –max-container-count is defined, the value of –max-memory-ratio will be ignored.

--no-rebalance-on-scale (= false) A boolean flag indicating whether containers should NOT be rebalanced after running an scale. The default behavior is to always rebalance the containers.

Remove an auto scale rule

```
$ tsuru node-autoscale-rule-remove [rule-name] [-y/--assume-yes]
```

Removes an auto-scale rule. The name of the rule may be omitted, which means "remove the default rule".

Flags:

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

Application Logging

Update logging configuration

```
$ tsuru docker-log-update [-r/--restart] [-p/--pool poolname] --log-driver <driver> [--log-opt name=v
```

Set custom configuration for container logs. By default tsuru configures application containers to send all logs to the tsuru/bs container through syslog.

Setting a custom log-driver allow users to change this behavior and make containers send their logs directly using the driver bypassing tsuru/bs completely. In this situation the 'tsuru app-log' command will not work anymore.

The –log-driver option accepts either the value 'bs' restoring tsuru default behavior or any log-driver supported by docker along with their –log-opt. See https://docs.docker.com/engine/reference/logging/overview/ for more details.

If –pool is specified the log-driver will only be used on containers started on the chosen pool.

Flags:

--log-driver (= "") Chosen log driver. Supported log drivers depend on the docker version running on nodes.

--log-opt (= {}) Log options send to the specified log-driver

-p, --pool (= "") Pool name where log options will be used.

-r, --restart (= false) Whether tsuru should restart all apps on the specified pool.

Show logging configuration

```
$ tsuru docker-log-info
```

Prints information about docker log configuration for each pool.

Quota management

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.

Change application quota

```
$ tsuru app-quota-change <app-name> <new-limit>
```

Changes the limit of units that an app can have.

The new limit must be an integer, it may also be “unlimited”.

Minimum # of arguments: 2

Change user quota

```
$ tsuru user-quota-change <user-email> <new-limit>
```

Changes the limit of apps that a user can create.

The new limit must be an integer, it may also be “unlimited”.

Minimum # of arguments: 2

View application quota

```
$ tsuru app-quota-view <app-name>
```

Displays the current usage and limit of the given app.

Minimum # of arguments: 1

View user quota

```
$ tsuru user-quota-view <user-email>
```

Displays the current usage and limit of the user.

Minimum # of arguments: 1

Other commands

Unlock an application

```
$ tsuru app-unlock -a <app-name> [-y]
```

Forces the removal of an application lock. Use with caution, removing an active lock may cause inconsistencies.

Flags:

-a, --app	(= “”) The name of the app.
-y, --assume-yes	(= false) Don’t ask for confirmation.

List tags with their associated apps and service instances

```
$ tsuru tag-list
```

Retrieves and shows a list of tags with the respective apps and service instances.

Installer

Install Tsuru and it's components

```
$ tsuru install-create [--config/-c config_file]
```

Installs Tsuru and It's components as containers on hosts provisioned with docker machine drivers.

The `--config` parameter is the path to a .yml file containing the installation configuration. If not provided, Tsuru will be installed into a VirtualBox VM for experimentation.

Flags:

-c, --config	(= "") Configuration file
-e, --compose	(= "") Components docker-compose file

List hosts created by the installer

```
$ tsuru install-host-list
```

List hosts created and registered by the installer.

SSH into an host created by the installer

```
$ tsuru install-ssh <hostname> [arg...]
```

Log into or run a command on a host with SSH.

Minimum # of arguments: 1

Uninstall Tsuru and it's components

```
$ tsuru install-remove [name] [-y/--assume-yes]
```

Uninstalls Tsuru and It's components.

Flags:

-c, --config	(= "") Configuration file
-y, --assume-yes	(= false) Don't ask for confirmation.

Help

Display all available commands

```
$ tsuru command [args]
```