TensorForce Documentation

Release 0.1.0a5

reinforce.io

Mar 07, 2018

Contents:

1 Installation **3**
- 1.1 Your Google Account (Including Service Account) . . . 3
- 1.2 Get the Google Cloud SDK (all platforms) . . . 5
- 1.3 Install Python3.5 or higher . . . 9
- 1.4 Get the TensorForce Client . . . 10
- 1.5 Setting an alias . . . 10

2 Usage - How to Run RL-Experiments in the Cloud? **11**
- 2.1 Creating the Cluster . . . 11

3 Internals - How does TensorForce-Client Work? **13**
- 3.1 The 2 base tools: gcloud and kubectl . . . 13
- 3.2 So how does it work now - really? . . . 14

4 Command Reference **15**
- 4.1 Projects . . . 15
- 4.2 Cluster . . . 15
- 4.3 Experiments . . . 16

5 Package/Class Reference **19**
- 5.1 Submodules . . . 19
- 5.2 Command Functions . . . 22

6 More information **25**

Python Module Index **27**

TensorForce-Client is an easy to use command line interface to the open source reinforcement learning (RL) library "TensorForce". This client helps you to setup and run your own RL experiments in the cloud (only google cloud supported so far), utilizing GPUs, multi-parallel execution algorithms, and TensorForce's support for a large variety of environments ranging from simple grid-worlds to Atari games and Unreal Engine 4 games. Tensorforce-client submits RL-jobs using Kubernetes and docker containers in a fully automated fashion. To read more about the internal workings of the client, check out this chapter here.

CHAPTER 1

Installation

In the following, we will be walking through the requirements to get you setup with tensorforce-client and how to satisfy or install each one of them.

1.1 Your Google Account (Including Service Account)

Create a new google account or use an existing one. Then go to http://console.cloud.google.com and create a new cloud project:

Fig. 1.1: Click here . . .

Enter a project name (only using alphanumeric or hyphens), for example: "TensorForce-Client". Select this project to be your active project from here on.

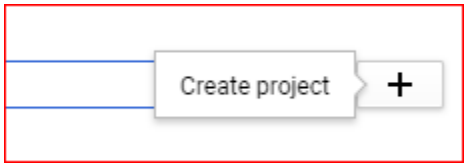


Fig. 1.2: . . . enter a name and click there.

Next, you will have to enable billing by providing Google with a (non-prepaid) credit card or a bank account number. This is necessary in order for Google to charge you for your future accrued cloud computing costs.

Then go to "APIs and Services" from the main menu (see image below) . . .

. . . and activate the following APIs (some of which may have already be enabled by default):

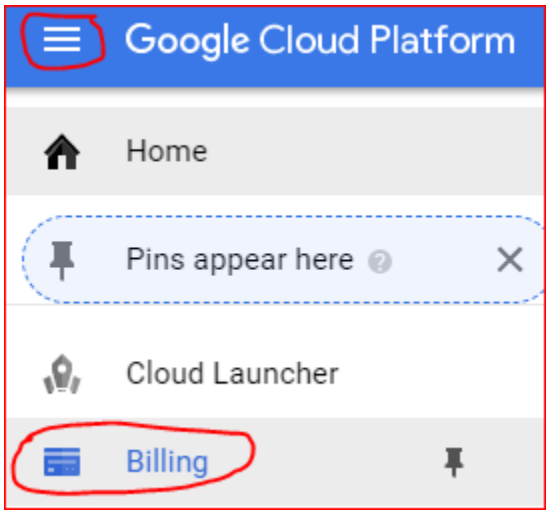


Fig. 1.3: How to get to "Manage your billing account".

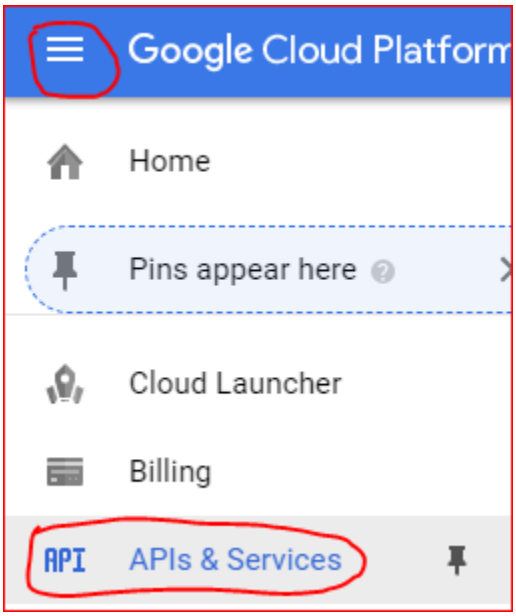


Fig. 1.4: How to get to "APIs and Services".

- Google Compute Engine API
- Google Kubernetes Engine API
- Google Cloud Storage
- Google Container Registry API
- Cloud Container Builder API

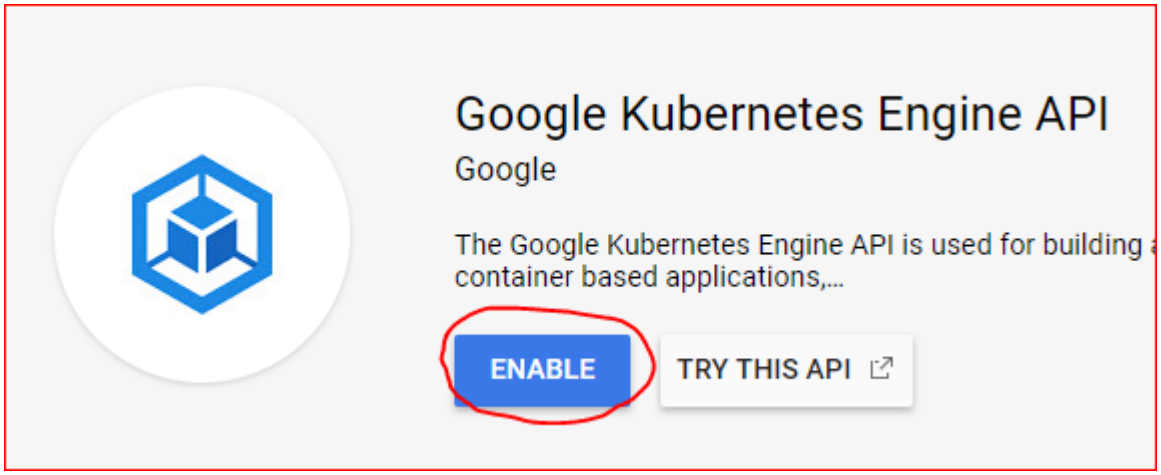


Fig. 1.5: Enabling the Kubernetes Engine API.

The last thing, we need to do with our google account is to add a so called "service account" to our project. This is so that we register our local glcoud app (see below) with our account and give permissions to it to create clusters and run experiments.

Go to the main menu -> "IAM & admin" -> "Service accounts":

... and click on:

Fill out the upcoming form like this (important to ask google to create a new private key for you, unless you would like to use an existing one):

... then click on "create" and store the upcoming json private key file somewhere on your local drive. Select the new service account in the list of accounts showing in your browser now and then click on "+Permissions" above. Then enter the name of the new service account ("test-account" in our example), and add the "Service Account User" permission to it:

When you *init* a new tensorforce-client project, you will be asked for the name of this new service account as well as for the location of the private key file on your local drive.

Congratulations! Your google account is now all set up to run cloud experiments with tensorforce-client. Time to install the remaining few pieces of software on our local machine.

1.2 Get the Google Cloud SDK (all platforms)

Go to the following URL and download the google SDK executable for your platform:

https://cloud.google.com/sdk/docs/quickstarts

Run the executable and follow the installation instructions.

Make sure you enable beta commands during installation as sometimes the tensorforce-client will rely on those.

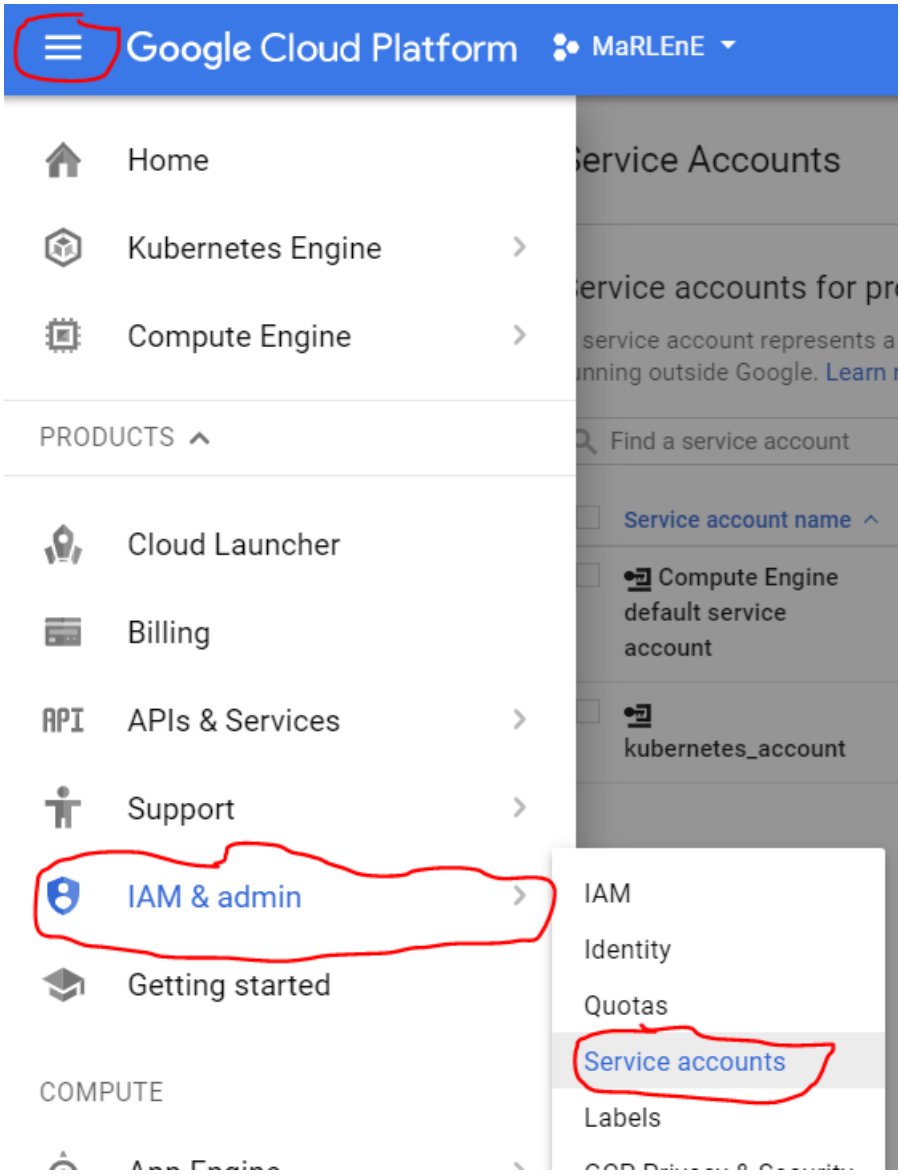


Fig. 1.6: How to get to "Service Accounts"

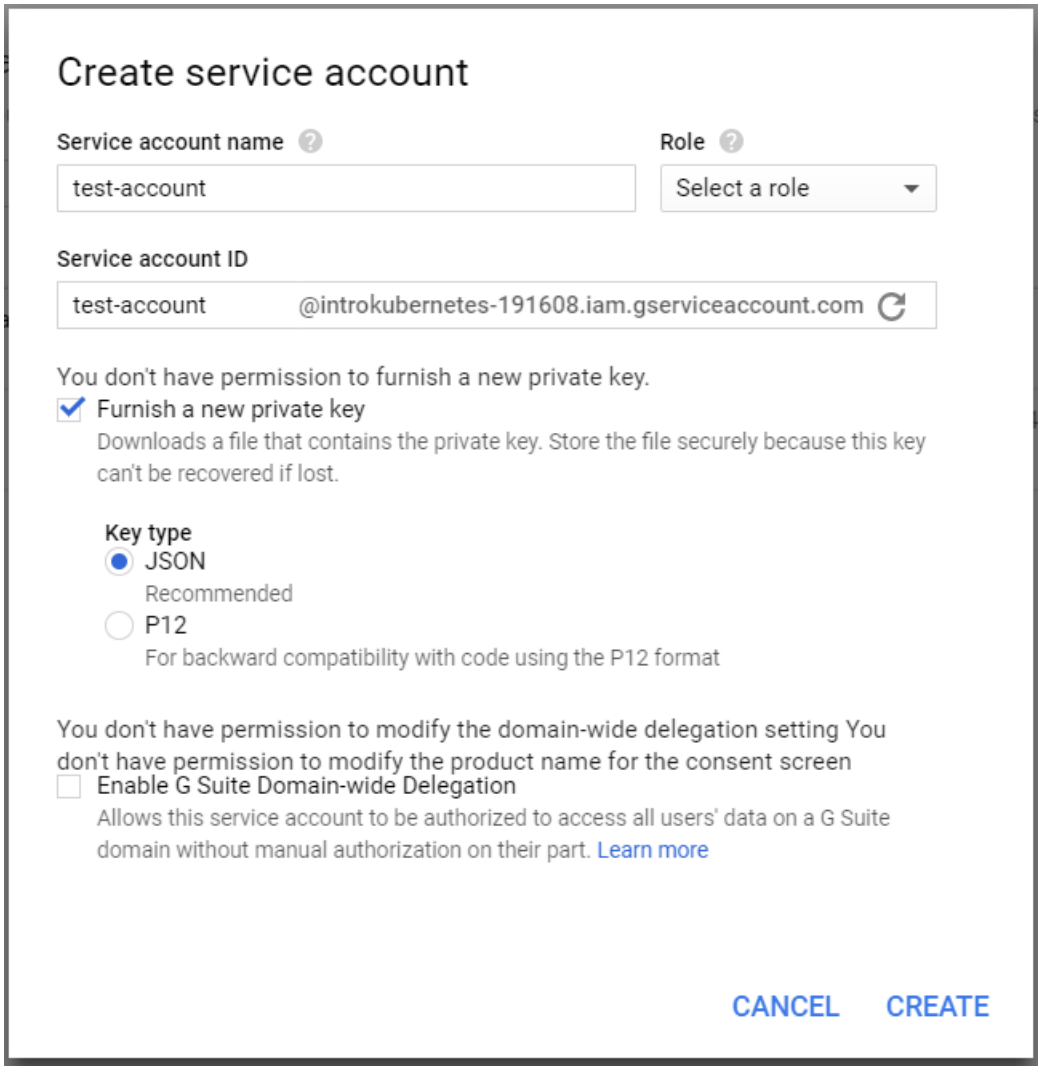
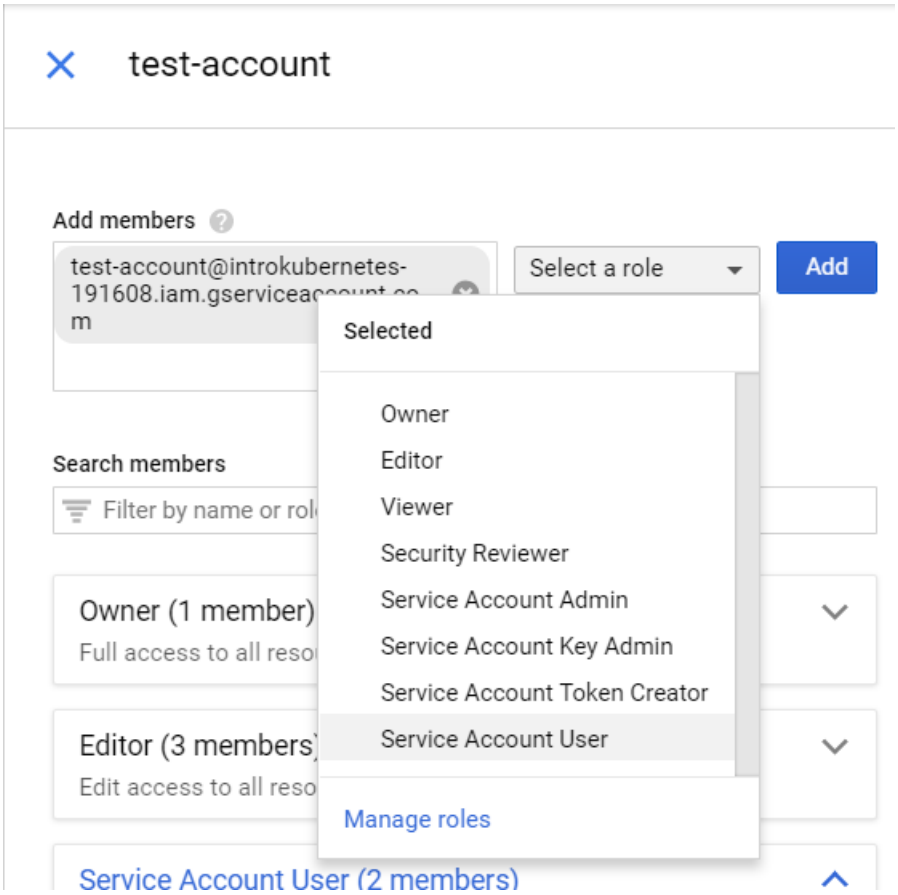


Fig. 1.7: Fill out the details for your new service account.


test-account
Add members
test-account@introkubernetes-
191608.iam.gserviceaccount.co
m
Select a role
Add
Selected
Owner
Editor
Viewer
Security Reviewer
Service Account Admin
Service Account Key Admin
Service Account Token Creator
Service Account User
Manage roles
Search members
Filter by name or rol
Owner (1 member)
Full access to all reso
Editor (3 members)
Edit access to all reso
Service Account User (2 members)

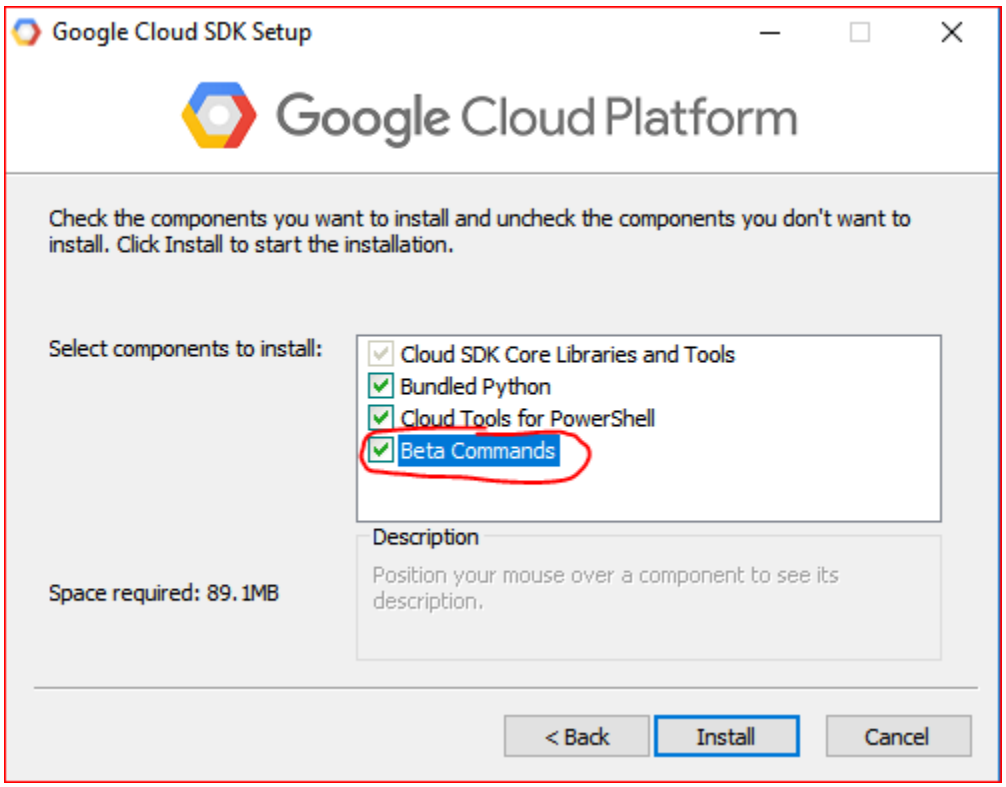


Fig. 1.8: Enable beta SDK commands during the google SDK installation procedure.

At the end of the installation process, depending on the installer version, say 'yes' to setting up the command line client (*gcloud*) or make sure that *gcloud init* runs. This will conveniently point you to your google account and (newly created) google cloud project and set some useful defaults (default compute region and zone).

There is a shell that comes with the installation. However, in order to run the tensorforce-client from within any shell (anaconda, git, or vanilla PowerShell), simply make sure that the gcloud installation path + */bin* is part of the %PATH env variable.

To make sure everything is setup correctly, run a test gcloud command. The output should look somewhat liks this:

```
$ gcloud version
Google Cloud SDK 187.0.0
alpha 2017.09.15
beta 2017.09.15
bq 2.0.28
core 2018.01.28
gsutil 4.28
```

1.3 Install Python3.5 or higher

. . . if you haven't already done so a long time ago ;-)

1.4 Get the TensorForce Client

The tensorforce-client is a python module that can be installed in one of two ways:

1.4.1 1) The easy way: pip Installation

Installing tensorforce-client through pip:

```
$ pip install tensorforce-client
```

Note: tensorforce-client neither needs the tensorforce library itself nor any of its core dependencies (e.g. tensorflow or tensorflow-gpu). So this should be an easy ride. Exception: If you would like to view (offline) tensorboard summaries of your models to be able to understand how well the algorithm is learning or to debug your models, you will have to install tensorflow like so:

```
$ pip install tensorflow
```

1.4.2 2) The hard way: git clone + setup.py

You can also get the latest development version of tensorforce-client by cloning/pulling it directly from our github repo and then running setup.py:

```
$ git clone github.com/reinforceio/tensorforce-client
$ cd tensorforce-client
$ python setup.py
```

1.5 Setting an alias

Tensorforce-client is a python module that should be run using:

```
$ tfcli [some command(s)] [some options]?
```

tfcli is a short for *python -m tensorforce_client*. Should the alias *tfcli* - for some reason - not work on the command line in your shell, you can set the alias manually for your current session as follows:

- Windows:

```
$ doskey tfcli=python -m tensorforce_client $*
```

- Linux:

```
$ alias tfcli='python -m tensorforce_client'
```

CHAPTER 2

Usage - How to Run RL-Experiments in the Cloud?

In this chapter, we will walk through an example experiment that runs the Atari 2600 game Space Invaders in an A3C fashion on a 2-node GPU cluster. We will use only 1 parameter server and 4 workers (agents that explore the space invaders environment). We will run a simple DQN algorithm for 1,000,000 timesteps (in total across all agents) using a replay buffer that fits 10,000 experience tuples (state, action, reward, next-state). The exploration strategy will be an epsilon decay starting from 1.0 at timestep 0 and linearly decreasing epsilon until it reaches 0.1 at timestep 1,000,000.

2.1 Creating the Cluster

The *tfcli cluster create* command starts a cluster in the cloud. For our example Use the

```
$ tfcli cluster create
```

command to start a cluster in the cloud.

For our example, we will use this command as follows:

The following options are supported:

CHAPTER 3

Internals - How does TensorForce-Client Work?

Tensorforce-client is under development and in alpha phase. As us developers will start running our own experiments using this client, we will add more and more functionality to the tool. In order to use tensorforce-client, you don't really need to know how it works internally. However, it is always beneficial to understand what's going on under the hood in case you run into problems or errors (which you should then report via the "Issues"-tab of the github repo).

3.1 The 2 base tools: gcloud and kubectl

Tensorforce-client uses two basic command line tools that you need to install on your local machine prior to running the client. These are *gcloud* and *kubectl* and they both come with the installation of the google SDK, which is described in the chapter on the installation procedure here.

3.1.1 Gcloud

Gcloud controls everything related to the google compute engine and the google Kubernetes engine. This means, it is responsible for:

- Creating and deleting Kubernetes (k8s) clusters of various sizes and types.
- Copying (via scp) config files and python scripts to all the nodes of a cluster.
- Executing (via ssh) commands on each node of a cluster.
- Downloading (via scp) results from an experiment from certain nodes of a cluster.

All the above tasks are done for you under the hood and you should never have to run a *gcloud* command manually.

3.1.2 Kubectl

Kubectl controls the workloads that go into the Kubernetes (k8s) service that's pre-installed on a cluster whenever you create one, using the *tfcli cluster create*-command of this client. Tensorforce-client makes sure all k8s yaml-config files are created correctly according to your experiment settings and runs the repsective kubectl commands automatically.

Just like for *gcloud*, all kubectl-specific tasks are done for you under the hood and you should never have to run a kubectl command manually.

3.2 So how does it work now - really?

When you start an experiment, tensorflow-client first checks the cluster setting on your command line or in the experiment's json and creates the cluster (via gcloud), if it's not already up.

Then it creates a k8s config file (yaml) out of a jinja2 template file, which will contain all the specifications for Kubernetes *pods*, *jobs*, *services* and *volume mounts* that are necessary to place the experiment in the Kubernetes engine running on the cluster.

All experiments are always run under Kubernetes using our docker container image. This image is hosted on dockerhub under ducandu/tfcli_experiment:[cpu|gpu].

Depending on the "run_mode" parameter of the experiment (settable through the experiment-json field: "run_mode"), the Kubernetes workloads will be set up as follows:

3.2.1 run_mode='single':

A single Kubernetes Pod is created on one node (you will get a warning if you use this mode on a cluster with more than one node) that runs our docker container (the GPU or the CPU version depending on whether your cluster has GPUs). Also within this container, a single RL environment (e.g. an Atari game instance) and a single agent with one tensorflow model is created and does all exploration in this environment.

3.2.2 run_mode='multi-threaded':

A single Pod is created on one node (you will get a warning if you use this mode on a cluster with more than one node) that runs our docker container (the GPU or the CPU version depending on whether your cluster has GPUs). The experiment's parameter *num_workers* determines how many parallel environment/agent-pairs are created inside that container and are being run each in a separate thread. Independent of the number of environment/agent-pairs (*num_workers*), only a single central tensorflow model is created and trained using so-called hogwild updates (see this paper here). The reinforcement learning algorithm that utilizes this technique was first published here.

3.2.3 run_mode='distributed':

The distributed uses many tensorflow models, where some serve merely as storages for the parameters (parameter-servers) and others are actively being trained (workers) and then send their gradients upstream to these parameter-servers. Each parameter server and each worker get their own Pod and all communication between them happens over the network. The number of k8s Pods created is hence the sum of the two experiment parameters *num_workers* and *num_parameter_servers*. Each Pod should preferably run on a separate node, but this is not a hard requirement (Kubernetes will figure out a good solution if you have fewer nodes in your cluster). Each of the "worker" Pods actively runs a single environment/agent-pair that owns its own tensorflow model and actively explores the environment. Parameter-servers also get an agent (including a model) but do not perform any exploration in an environment. Paraeter servers are only there for receiving gradient updates from workers and then redistributing these updates back to all other workers. This procedure utilizes tensorflow's built-in distributed learning architecture. Experiments with run-mode "distributed" follow the procedure described first in this paper here.

CHAPTER 4

Command Reference

4.1 Projects

Projects in tensorforce-client are practically directories on your local drive that are linked to arbitrary google cloud projects from your google account. It's ok to link more than one local project to the same remote (cloud) project.

To make a local directory a tensorforce-client project, cd into that directory and run:

```
$ tfcli init -r [gcloud remote project ID] -n [local project name]
```

If you don't specify a remote project ID, the client will ask you to provide one. You have to link your local project to an already existing remote google cloud project. The project's name (*-n* option) is non-mandatory and the tool will use the remote project's name.

You will also be asked to provide the service-account (see installation instructions here) that you would like to use for running *tfcli* commands as well as the private key file to use for client-server communications.

Each local tensorforce-client project contains a convenience directory for configuration files (*config/*), into which the tool copies - upon project creation - all tensorforce default config files for clusters, experiments, agents, networks, and environments. You can either add more config files to this directory, change existing ones, or ignore this folder altogether and create and maintain your config files elsewhere.

4.2 Cluster

4.2.1 cluster –help

Lists all sub-commands that the main *cluster* command supports.

4.2.2 cluster create

```
$ tfcli cluster create -f [json filename] -n [give the cluster some name]
```

Creates a Kubernetes (k8s) cluster in the cloud.

The two most important command line flags are *-f [json file]* and *-n [name for the cluster]*. Take a look in the *configs/clusters* directory that is created automatically inside each new project for some example cluster setups (including GPU clusters). For the json-file, you don't need to give a path (nor the .json extension) if the file is located in the *configs/clusters* directory.

Instead of providing the *-f* flag, however, you could also define your cluster's parameters entirely on the command line (thus without the need for a json file). For supported flags, run:

```
$ tfcli cluster create --help
```

For an explanation of the (more powerful) supported json fields, take a look at the cluster class reference here.

4.2.3 cluster list

```
$ tfcli cluster list
```

Lists all clusters currently running in the cloud.

4.2.4 cluster delete

Deletes (shuts down) a cluster running in the cloud. You will only have to provide the name tat you gave to the cluster when you created it via the *-c* flag.

```
$ tfcli cluster delete -c [cluster's name]
```

Use the *cluster list* command to get the names for all currently running clusters.

Also, if you simply would like to shut down all currently running clusters in your associated gcloud project, do:

```
$ tfcli cluster delete --all
```

4.3 Experiments

4.3.1 experiment –help

Lists all sub-commands that the main *experiment* command supports.

4.3.2 experiment new

```
$ tfcli experiment new -f [json filename] -n [give the experiment some name]
```

Creates a local(!) experiment entry in the current project. By default, this command does not start the experiment in the cloud.

The two most important command line flags are *-f [json file]* and *-n [name for the experiment]*. Take a look in the *configs/experiments* directory that is created automatically inside each new project for some example experiment setups. For the json-file, you don't need to give a path (nor the .json extension) if the file is located in the *configs/experiments* directory.

The main parts of an RL experiment are generally the algorithm used (set via the json *agent* setting), the environment (set via athe json *environment* setting), in which the agent acts, the neural-network that the agent uses (set via the json *network* setting), the length of the experiment (set via the various num_timesteps, num_episodes, etc.. settings), and the run_mode. For a detailed description of the different run_modes, take a look here.

For an explanation of all supported json fields, take a look at the Experiment class reference here.

Instead of providing the *-f* flag on the command line, you could also define your experiment's parameters entirely on the command line (thus without the need for a json file). For supported flags, run:

```
$ tfcli experiment new --help
```

4.3.3 experiment list

```
$ tfcli experiment list
```

Lists all experiments that currently exist in this local project. Their run status ('running', 'paused', etc..) is shown as well.

4.3.4 experiment start

```
$ tfcli experiment start -e [name of the experiment to start] -c [name of the cluster␣
↪to start it on]
```

Starts an already existing experiment on a given cluster in the cloud. The *-c* option is optional and can be omitted, because all experiments are created automatically with a cluster specification. Should the cluster (the *-c* provided one or the one that's coming with the experiment) not already be running, it will be created (started) in the cloud before the experiment is set up.

A started experiment will run until one of the stop conditions (settable through *num_episodes* and/or *num_timesteps* in the experiment's json spec file or via the command line) is met. If the cluster the eperiment runs on belongs to the experiment (as opposed to a separately created cluster), that cluster is shut down after experiment completion.

> **NOTE: Tensorforce-client is still largely under development. As we are conducting reinforcement learning experiments with our different TensorForce-supported environments in the cloud, we will add more and more functionality to this client, especially focusing on representing results and making it easier to benchmark and compare different algorithms and neural network models. The following experiment-related sub-commands are not implemented yet:**

4.3.5 experiment pause

Pauses an already running experiment. A paused experiment can be resumed by passing the ''

4.3.6 experiment stop

Stops (aborts) an experiment

CHAPTER 5

Package/Class Reference

5.1 Submodules

Tensorforce-client only has two major submodules, which are Experiment and Cluster. Currently - as all interaction with the client happens from the command line - you won't need to know the details of those two classes. We may add support for python scripting against this client library in the future so you will be able to automate your experiments and cluster creation/deletion tasks.

5.1.1 Experiments: (tensorforce_client.experiment)

class tensorforce_client.experiment.**Experiment**(***kwargs*)

Bases: object

__init__(***kwargs*)

Keyword Arguments

- **file** (*str*) – The Experiment's json spec file (can contain all other args).
- **name** (*str*) – The name of the Experiment. This is also the name of the folder where it is stored.
- **environment** (*str*) – The filename of the json env-spec file to use (see TensorForce documentation).
- **agent** (*str*) – The filename of the json agent-spec file to use (see TensorForce documentation).
- **network** (*str*) – The filename of the json network-spec file to use (see TensorForce documentation).
- **cluster** (*str*) – The filename of the json cluster-spec file to use (see class Cluster).
- **episodes** (*int*) – The total number of episodes to run (all parallel agents).

- **total_timesteps** (*int*) – The max. total number of timesteps to run (all parallel agents).
- **max_timesteps_per_episode** (*int*) – The max. number of timesteps to run in each episode.
- **deterministic** (*bool*) – Whether to not(!) use stochastic exploration on top of plain action outputs.
- **repeat_actions** (*int*) – The number of actions to repeat for each action selection (by calling agent.act()).
- **debug_logging** (*bool*) – Whether to switch on debug logging (default: False).
- **run_mode** (*str*) – Which runner mode to use. Valid values are only 'single', 'multi-threaded' and 'distributed'.
- **num_workers** (*int*) – The number of worker processes to use (see `distributed` and `multi-threaded` run_modes).
- **num_parameter_servers** (*int*) – The number of parameter servers to use (see distributed tensorflow).
- **saver_frequency** (*str*) – The frequency with which to save the model. This is a combination of an int and a unit (e.g. "600s"), where unit can be "s" (seconds), "e" (episodes), or "t" (timesteps).
- **summary_frequency** (*str*) – The frequency with which to save a tensorboard summary. This is a combination of an int and a unit (e.g. "600s"), where unit can be "s" (seconds) or "t" (timesteps). The episode unit (e) is not allowed here.

download()
Downloads the experiment's results (model checkpoints and tensorboard summary files) so far.

generate_locally()
Writes the local json spec file for this Experiment object into the Experiment's dir. This file contains all settings (including agent, network, cluster, run-mode, etc..).

pause(*project_id*)
Pauses the already running Experiment.

Parameters **project_id** (*str*) – The remote gcloud project-ID.

setup_cluster(*cluster*, *project_id*, *start=False*)
Given a cluster name (or None) and a remote project-ID, sets up the cluster settings for this Experiment locally. Also starts the cluster if start is set to True.

Parameters

- **cluster** (*str*) – The name of the cluster. If None, will get cluster-spec from the Experiment, or create a default Cluster object.
- **project_id** (*str*) – The remote gcloud project ID.
- **start** (*bool*) – Whether to already create (start) the cluster in the cloud.

Returns: The Cluster object.

start(*project_id*, *resume=False*, *cluster=None*)
Starts the Experiment in the cloud (using kubectl). The respective cluster is started (if it's not already running).

Parameters

- **project_id** (*str*) – The remote gcloud project-ID.

- **resume** (*bool*) – Whether we are resuming an already started (and paused) experiment.
- **cluster** (*str*) – The name of the cluster to use (will be started if not already running). None for using the Experiment's own cluster or - if not given either - a default cluster.

stop(*no_download=False*)
Stops an already running Experiment by deleting the Kubernetes workload. If no_download is set to False (default), will download all results before stopping. If the cluster that the experiment runs on is dedicated to this experiment, will also delete the cluster.

Parameters **no_download** (*bool*) – Whether to not(!) download the experiment's results so far (default: False).

write_json_file(*file=None*)
Writes all the Experiment's settings to disk as a json file.

Parameters **file** (*str*) – The filename to use. If None, will use the Experiment's filename.

tensorforce_client.experiment.**get_experiment_from_string**(*experiment*, *running=False*)
Returns an Experiment object given a string of either a json file or a name of an already existing eperiment.

Parameters

- **experiment** (*str*) – The string to look for (either local json file or local experiment's name)
- **running** (*bool*) – Whether this experiment is already running.

Returns The found Experiment object.

tensorforce_client.experiment.**get_local_experiments**(*as_objects=False*)

Parameters **as_objects** (*bool*) – Whether to return a list of strings (names) or actual Experiment objects.

Returns: A list of all Experiment names/objects that already exist in this project.

5.1.2 Clusters: (tensorforce_client.cluster)

class tensorforce_client.cluster.**Cluster**(***kwargs*)
Bases: object

__init__(***kwargs*)
A cloud cluster object specifying things like: number of nodes, GPUs per node and GPU type, memory per node, disk size, zone, etc..

Parameters **kwargs** (*any*) – See below.

Keyword Arguments

- **file** (*str*) – The filename of a cluster spec json file to use. Single settings in this file can be overwritten by specifying these in further kwargs to this c'tor.
- **name** (*str*) – The name of the cluster.
- **machine_type** (*str*) – The machine type to use for all nodes in the cluster. Machine types can either be gcloud-accepted strings such as everything listed in `gcloud compute machine-types list` or custom strings that conform to these rules: https://cloud.google.com/compute/docs/instances/creating-instance-with-custom-machine-type. When the kwargs `cpus_per_node` and `memory_per_node` are given, tensorforce-client will automatically create the correct machine-type.

- **cpus_per_node** (*int*) – The number of vCPUs per node.
- **gpus_per_node** (*int*) – The number of (physical) GPUs per node.
- **gpu_type** (*str*) – The GPU type to use. Supported are only 'nvidia-tesla-k80' and 'nvidia-tesla-p100'.
- **memory_per_node** (*int*) – The memory (in Gb) per node.
- **num_nodes** (*int*) – The number of nodes for the cluster.
- **disk_size** (*int*) – The amount of disk space per node in Gb.
- **location** (*str*) – The location of the cluster. Default us the gcloud/project set default zone.

create()
Create the Kubernetes cluster with the options given in self. This also sets up the local kubectl app to point to the new cluster automatically.

delete()
Deletes (shuts down) this cluster in the cloud.

get_spec()
Returns: Dict of the important settings of this Cluster.

ssh_parallel(**items*, ***kwargs*)
Runs commands via ssh and/or scp commands on all nodes in the cluster in parallel using multiple threads.

Parameters

- **items** (*List[Union[str,tuple]]*) – List of commands to execute. Could be either of type str (ssh command) or a tuple/list of two items (`from` and `to`) for an scp command.
- **kwargs** (*any*) – silent (bool): Whether to execute all commands silently (default: True).

tensorforce_client.cluster.**get_cluster_from_string**(*cluster*, *running_clusters=None*)
Returns a Cluster object given a string of either a json file or an already running remote cluster's name.

Parameters

- **cluster** (*str*) – The string to look for (either local json file or remote cluster's name)
- **running_clusters** (*dict*) – Specs for already running cloud clusters by cluster name.

Returns The found Cluster object.

5.2 Command Functions

tensorforce_client.commands.**cmd_cluster_create**(*args*)

tensorforce_client.commands.**cmd_cluster_delete**(*args*)

tensorforce_client.commands.**cmd_cluster_list**()

tensorforce_client.commands.**cmd_experiment_download**(*args*)

tensorforce_client.commands.**cmd_experiment_list**()

tensorforce_client.commands.**cmd_experiment_new**(*args*, *project_id=None*)

tensorforce_client.commands.**cmd_experiment_pause**(*args*, *project_id*)

tensorforce_client.commands.**cmd_experiment_start**(*args*, *project_id*)

tensorforce_client.commands.**cmd_experiment_stop**(*args*)

tensorforce_client.commands.**cmd_init**(*args*)

CHAPTER 6

More information

You can find more information at our TensorForce-Client GitHub repository.

For the core TensorForce library, visit: https://github.com/reinforceio/TensorForce.

We also have a seperate repository available for benchmarking our algorithm implementations [here](https://github.com/reinforceio/tensorforce-benchmark).

Python Module Index

t

tensorforce_client.cluster, 21
tensorforce_client.commands, 22
tensorforce_client.experiment, 19

Index

Symbols

__init__() (tensorforce_client.cluster.Cluster method), 21
__init__() (tensorforce_client.experiment.Experiment method), 19

C

Cluster (class in tensorforce_client.cluster), 21
cmd_cluster_create() (in module tensorforce_client.commands), 22
cmd_cluster_delete() (in module tensorforce_client.commands), 22
cmd_cluster_list() (in module tensorforce_client.commands), 22
cmd_experiment_download() (in module tensorforce_client.commands), 22
cmd_experiment_list() (in module tensorforce_client.commands), 22
cmd_experiment_new() (in module tensorforce_client.commands), 22
cmd_experiment_pause() (in module tensorforce_client.commands), 22
cmd_experiment_start() (in module tensorforce_client.commands), 22
cmd_experiment_stop() (in module tensorforce_client.commands), 22
cmd_init() (in module tensorforce_client.commands), 23
create() (tensorforce_client.cluster.Cluster method), 22

D

delete() (tensorforce_client.cluster.Cluster method), 22
download() (tensorforce_client.experiment.Experiment method), 20

E

Experiment (class in tensorforce_client.experiment), 19

G

generate_locally() (tensorforce_client.experiment.Experiment method), 20
get_cluster_from_string() (in module tensorforce_client.cluster), 22
get_experiment_from_string() (in module tensorforce_client.experiment), 21
get_local_experiments() (in module tensorforce_client.experiment), 21
get_spec() (tensorforce_client.cluster.Cluster method), 22

P

pause() (tensorforce_client.experiment.Experiment method), 20

S

setup_cluster() (tensorforce_client.experiment.Experiment method), 20
ssh_parallel() (tensorforce_client.cluster.Cluster method), 22
start() (tensorforce_client.experiment.Experiment method), 20
stop() (tensorforce_client.experiment.Experiment method), 21

T

tensorforce_client.cluster (module), 21
tensorforce_client.commands (module), 22
tensorforce_client.experiment (module), 19

W

write_json_file() (tensorforce_client.experiment.Experiment method), 21