Stackviz

Release 0.0.1.dev353

OpenStack Foundation

May 02, 2024

CONTENTS

CHAPTER ONE

OVERVIEW

Stackviz is a visualization utility to help analyze the performance of DevStack setup and Tempest test runs. The following documentation details the procedures for installing Stackviz on your local machine to view local runs, as well as how to use Stackviz to view upstream runs. For more information on how Stackviz runs, see the manual pages.

CHAPTER

TWO

LOCAL STACKVIZ

2.1 Team and repository tags

openstack community project cii best practices passing

2.2 StackViz

A visualization utility to help analyze the performance of DevStack setup and Tempest executions. This repository can be cloned and built to use Stackviz with local run data. Stackviz is currently in the process of being implemented upstream (see Roadmap and Planning). To use Stackviz with upstream gate runs, please see the server deployment project at:

- https://github.com/timothyb89/stackviz-deployer

2.2.1 Installation

Installation - Frontend

Installation of the frontend requires Node.js and Gulp. On Ubuntu:

```
sudo apt-get install nodejs
sudo apt-get install npm
sudo npm install -g gulp
```

Then, install the Node modules by running, from the project directory:

```
npm install
```

Installation - Processing

The data processor is a small Python module located in the same source tree. To install, run:

```
sudo pip install .
```

2.2.2 Usage

Usage - Development

A development server can be run as follows:

```
gulp dev
```

This will open a web browser and reload code automatically as it changes on the filesystem.

If you have subunit and dstat logs, you can create a config.json to display your runs:

```
stackviz-export -f <path/to/subunit> --dstat <path/to/dstat> app/data/
```

During `gulp dev`, files written to `app/data/` will be automatically synchronized with the browser. Note that these files will *not* be copied to `build/` during `gulp prod`, but you can copy them manually using `gulp data`.

Usage - Production

The production application can be build using:

```
gulp prod
```

This will automatically build portable html/javascript and python utilities into `dist/stackviz-VERSION.tar.gz`.

You should probably install this into a `virtualenv` on the target system:

```
virtualenv stackviz
./virtualenv/bin/pip install /path/to/stackviz-VERSION.tar.gz
# to run stackviz export
./virtualenv/bin/stackviz-export
```

Note the required html will be placed in `virtualenv/share/stackviz-html` as a data-file (or elsewhere, if installed as a system package; this may vary on distributions). This can be moved as required. Note that all files in there are not required:

- Directory structure (`js/`, `css/`, `fonts/`, `images/`): required.
- Static resources (`fonts/`, `images/`): required.
- Core files (`index.html`, `js/main.js`, `css/main.css`): required unless gzipped versions are used.
- Gzipped versions of core files (`*.gz`): not required, but preferred. Use instead of plain core files to save on disk usage and bandwidth.

- Source maps (`js/main.js.map`, `js/main.js.map.gz`): only required for debugging purposes.

Data should be written to `stackviz-html/data/` using `stackviz-export` like above.

2.2.3 Testing

- Python tests: `tox -e py36`
- JavaScript unit tests: `gulp unit`
- JavaScript E2E tests: `gulp e2e`

2.2.4 Manuals & Developer Docs

For more detailed information on how Stackviz works, please see the manuals located at doc/source/man/

2.2.5 Roadmap and Planning

- Planning: https://etherpad.openstack.org/p/stackviz
- Gate integration planning: https://etherpad.openstack.org/p/BKgWlKIjgQ

2.3 Installation

2.3.1 Installation - Frontend

Installation of the frontend requires Node.js and Gulp. On Ubuntu:

```
sudo apt-get install nodejs npm nodejs-legacy
sudo npm install -g gulp
```

Then, install the Node modules by running, from the project directory:

```
npm install
```

2.3.2 Installation - Processing

The data processor is a small Python module located in the same source tree. To install, run:

```
sudo pip install .
```

2.4 Usage

2.4.1 Usage - Development

A development server can be run as follows:

```
gulp dev
```

This will open a web browser and reload code automatically as it changes on the filesystem.

If you have subunit and dstat logs, you can create a config.json to display your runs:

```
stackviz-export -f <path/to/subunit> --dstat <path/to/dstat> app/data/
```

During `gulp dev`, files written to `app/data/` will be automatically synchronized with the browser. Note that these files will *not* be copied to `build/` during `gulp prod`, but you can copy them manually using `gulp data`.

2.4.2 Usage - Production

The production application can be build using:

```
gulp prod
```

The result will be written to `./build` and should be appropriate for distribution. Note that all files are not required:

- Directory structure (`js/`, `css/`, `fonts/`, `images/`): required.
- Static resources (`fonts/`, `images/`): required.
- Core files (`index.html`, `js/main.js`, `css/main.css`): required unless gzipped versions are used.
- Gzipped versions of core files (`*.gz`): not required, but preferred. Use instead of plain core files to save on disk usage and bandwidth.
- Source maps (`js/main.js.map`, `js/main.js.map.gz`): only required for debugging purposes.

Data should be written to `build/data/` using `stackviz-export` like above. Note that the static production code generated above is portable, and can be generated anywhere and copied to another host to be combined with exported data.

CHAPTER THREE

FOR CONTRIBUTOR

If you are a new contributor to Stackviz please refer: *So You Want to Contribute*

3.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Stackviz.

3.1.1 Communication

- IRC channel `#openstack-qa` at OFTC
- Mailing list (prefix subjects with `[qa]` for faster responses) http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

3.1.2 Contacting the Core Team

Please refer to the Stackviz Core Team contacts.

3.1.3 New Feature Planning

If you want to propose a new feature please read Feature Proposal Process

3.1.4 Task Tracking

There is no separate task tracking tool for Stackviz, we track our tasks in Launchpad.

3.1.5 Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Launchpad. There is no separate Launchpad for Stackviz. More info about Launchpad usage can be found on OpenStack docs page

3.1.6 Getting Your Patch Merged

All changes proposed to the Stackviz requires single `Code-Review +2` votes as minimum from Stackviz core reviewers who can approve patch by giving `Workflow +1` vote.

3.1.7 Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

The Release Process for QA is documented in QA Release Process.

CHAPTER

FOUR

MANUAL PAGES

4.1 Welcome to Stackvizs Manuals!

In this directory, you will find detailed documentation describing Stackviz components and how they work together.

Stackviz is broken up into two distinct components: a Python processing module (stackviz/stackviz) and an AngularJS front-end (stackviz/app). Stackviz also uses Gulp to manage various tasks including building sites and running tests. For information on each of these components, see their corresponding RST entry. Below is a listing of each major subdirectory in Stackviz.

4.1.1 Directories:

- `./app/`: AngularJS front-end.
- `./doc/`: Stackvizs documentation.
- `./gulp/`: Gulp used for task management.
- `./stackviz/`: Python processing module.
- `./test/`: Unit and e2e tests.

Documentation for the Python processing module and AngularJS front-end:

Python Data-Processing Module (stackviz-export)

The main purpose of `stackviz-export` is to parse subunit and dstat logs in order to generate configuration files for the AngularJS front-end.

Installation

Once Stackviz has been cloned into a suitable directory, setting up the module is as simple as:

```
sudo pip install .
```

Usage

`stackviz-export [options] <DEST>`

Where DEST is the output directory of the module. If DEST does not exist, a new directory will be created. One of the following input options must be chosen:

-f, stream-file FILE
Specifies a subunit stream file to be used with the exporter. This argument can be used multiple times to specify additional subunit files.

-i, stdin
Instructs stackviz-export to read a subunit stream from stdin.

-r, repository REPOSITORY
Specifies a `.stestr` to read subunit streams from. This argument can be used multiple times to specify additional repositories.

Stackviz also visualizes machine utilization statistics using dstat. To attach a dstat.csv log to the subunit output, specify the following option:

dstat FILE
Specifies a csv-formatted dstat log file that corresponds with the provided subunit stream file.

Additional options:

-h help
Print help message.

-z gzip
Enables gzip compression for data files.

Output

`stackviz-export` outputs the following files to the destination directory. Note that <source> in the details, raw, and tree logs refer to what stream source the

config.json
Contains all the basic information about a dataset that the front-end needs. There will be one *tempest* entry for every dataset that was generated in `stackviz-export`. Each *tempest* entry has general information about each run, as well as the locations of the details, raw, and tree JSON files.

dstat_log.csv
This file will only be present if a dstat log was used in the corresponding `stackviz-export` run. Has a wide variety of system statistics including CPU, memory, and disk utilization. This information is displayed on the timeline graph.

tempest_<source>_<id>_details.json
The details log contains timestamp and status information for tests in addition to all of the logs associated with the test (e.g. tracebacks). These artifacts are displayed in the test details page.

tempest_<source>_<id>_raw.json

Contains nearly all information available about tests:

- `status`: pass, fail, or skipped
- `name`: full name of test

- `tags`: which worker the test was run on
- `details`: empty, this info is available in the details JSON
- `duration`: how long the test took, in seconds
- `timestamps`: timestamps at test begin and test end

This file is used in the timeline and test details page.

tempest_<source>_<id>_tree.json
Stores test names in a hierarchy for display on the deprecated sunburst diagram. Not currently used by any page in Stackviz.

AngularJS Front-end

The AngularJS front-end uses config files generated by `stackviz-export` to display a variety of information regarding test runs. This document breaks down the various components of the Stackviz Angular app.

Pages

Home

Path
<host>/#/

Directive
`./app/views/home.html`

Controller
`./app/js/controllers/home.js`

The landing page for Stackviz consists of two elements. The first is a summary panel that shows statistics for the run including runtime (MM:SS), total number of tests run, number of failed tests, and number of skipped tests. The button in the footer of this panel links to the timeline, where individual tests can be browsed further. The second element is a panel showing all of the failures for the current run, including the last few lines of their tracebacks. The test divs here link to their corresponding Test Details page.

Timeline

Path
<host>/#/<run>/timeline?test=<test>/

Directive
`./app/views/timeline.html`

Controller
`./app/js/controllers/timeline.js`

The Timeline provides an overview of all the tests that were executed as part of a run. Each lane in the timeline corresponds to one worker thread on the host machine. Each rectangle in the timeline represents one test. When a rectangle is selected, the details panel below the timeline is populated with information about the test. Each test rectangle is also color-coordinated: green is passing, blue is skipped, and red is failed.

The details panel below the timeline shows information pertaining to the current test (it is empty if no test is selected): test class, test module, which worker executed it, the duration, and start & finish timestamps. The footer contains a button that links to the test details page for the selected test.

Test Details

Path
: <host>/#/<run>/test-details/<test>/

Directive
: `./app/views/test-details.html`

Controller
: `./app/js/controllers/test-details.js`

The test details page consists of one panel that displays various log info from one test. The first tab contains summary information similar to the info found on the test panel on the timeline. Additional tabs in this panel are dependent upon the logs that the test kept. Each of these tabs provides additional information to aid debugging. The most common tabs include:

pythonlogging
: Contains logs for API calls that were used in the test. This log is often quite large, as it contains full headers for every request at INFO, DEBUG, WARNING, and ERROR levels. To make searching these logs easier, the test details page has a built in filter for parsing by log level. In the header of the test details page, the magnifying glass can be clicked to only show pythonlogging lines that correspond to a certain level of detail. To find errors in pythonlogging quickly, it is advisable to only select the WARNING and ERROR levels for display.

reason
: Only available for skipped tests. Lists the reason for skipping the test, usually to avoid triggering an outstanding bug.

traceback
: Only available for failed tests. Shows the full traceback of the test runners error output when the test failed. This is useful in quickly isolating the cause of a failure. There can be multiple traceback logs (e.g. traceback, traceback1) for one test.

When enough information has been gleaned from more detailed logs, the button in the panel filter can be used to quickly navigate back to the timeline page.

Directives

tempestSummary
: The tempest summary directive consists of one panel that shows stats for one run: Duration of the run, number of tests run, number of tests skipped, and number of tests failed. `timeDiff` (the duration of the run) is calculated from the start and end timestamps contained in summary data. All other fields are populated directly from the summary data, via a call to the dataset service.

testDetailsSearch
: `testDetailsSearch` uses two HTML pages to search the test details page: `test-details-search-popover.html` and `test-details-search.html`. The popover

contains the filter levels for the `pythonlogging` tab: INFO, DEBUG, WARNING, ERROR. This directive is used as the template for `test-details-search`, per AngularJS popover convention. The function used to parse the logs, `parsePythonLogging` actually lives in the controller for testDetailsSearch, and is passed through both the prior directives scopes. This function reads in the `pythonLogging` tab as one text object, then splits it by `n` to create an array of lines. Each line is then added back to the `pythonLogging` tab if it contains the specific log level somewhere in the line.

timeline
The timeline directive is a container for the actual timeline components, detailed below.

timelineDetails

timelineDstat

timelineOverview

timelineSearch

timelineViewport

Services

dataset
The dataset service is an API that provides the front-end with all of the data generated by `stackviz-export`. All data processed by `stackviz-export` ends up in the *./app/data/* directory to be called by dataset service with `$http` and `$q` directives. Below is the list of calls:

- `list` returns *config.json* using GET.
- `get(id)` calls `list`, then iterates through all the available datasets for the requested id number. Rejects if not found.
- `raw(dataset)` returns *<dataset>_raw.json* file using GET.
- `details(dataset)` returns *<dataset>_details.json* file using GET.
- `tree(dataset)` returns *<dataset>_tree.json* file using GET.
- `dstat(dataset)` returns *dstat_log.csv* file using GET, if available.

progress
A wrapper for `nprogress`, a progress bar library. Used in the timeline and test details pages to show progress in loading datasets.

CHAPTER

FIVE

INDICES AND TABLES

- search