InfraScraper Documentation

Release 0.2.0

InfraScraper Team

Dec 24, 2017

Contents

CHAPTER 1

Application Overview

1.1 Installation

1.1.1 PIP Installation

Release version of infra-scraper is currently available on Pypi, to install it, simply execute:

```
pip install infra-scraper
```

1.1.2 Instalation from Source

To bootstrap latest development version into virtualenv, run following commands:

```
git clone git@github.com:cznewt/infra-scraper.git
cd infra-scraper
virtualenv venv
source venv/bin/activate
python setup.py install
```

1.2 Configuration

You provide one configuration file for all providers. The default location is `/etc/infra-scraper/config.yaml` but it can be overriden by `INFRA_SCRAPER_CONFIG_PATH` environmental variable, for example:

```
export INFRA_SCRAPER_CONFIG_PATH=~/scraper.yml
```

1.2.1 Configuration in ETCD

You can use ETCD as a storage backend for the configuration and scrape results. Following environmental parameters need to be set:

```
export INFRA_SCRAPER_CONFIG_BACKEND=etcd
export INFRA_SCRAPER_CONFIG_PATH=/service/scraper/config
```

1.2.2 Storage Configuration

You can set you local filesystem path where scraped data will be saved.

```
storage:
  backend: localfs
  path: /tmp/scraper
endpoints: {}
```

You can also set the scraping storage backend to use the ETCD service instead of a local filesystem backend.

```
storage:
  backend: etcd
  path: /scraper
endpoints: {}
```

1.2.3 Endpoints Configuration

Each endpoint kind expects a little different set of configuration. Look at individual chapters for samples of required parameters to setup individual endpoints.

1.3 Usage

The application comes with several entry commands:

1.3.1 Scraping Commands

scraper_get <endpoint-name>

Scrape single endpoint once.

scraper_get_forever <endpoint-name>

Scrape single endpoint continuously.

scraper_get_all

Scrape all defined endpoints once.

scraper_get_all_forever

Scrape all defined endpoints continuously.

1.3.2 UI and Utility Commands

scraper_status

Display the service status, endpoints, scrapes, etc.

scraper_web

Start the UI with visualization samples and API that provides the scraped data.

CHAPTER 2

Supported Platforms

2.1 Amazon Web Services

AWS scraping uses `boto3` high level AWS python SDK for accessing and manipulating AWS resources.

```
endpoints:
  aws-us-west-2-admin:
    kind: aws
    config:
      region: us-west-2
      aws_access_key_id: <access_key_id>
      aws_secret_access_key: <secret_access_key>
```

2.2 Kubernetes Clusters

Kubernetes requires some information from `kubeconfig` file. You provide the parameters of the cluster and the user to the scraper. These can be found under corresponding keys in the kubernetes configuration file.

```
endpoints:
  k8s-admin:
    kind: kubernetes
    layouts:
    - force
    - hive
    config:
      cluster:
        server: https://kubernetes-api:443
        certificate-authority-data: |
          <ca-for-server-and-clients>
      user:
        client-certificate-data: |
          <client-cert-public>
```

```
        client-key-data: |
          <client-cert-private>
```

2.3 OpenStack Clouds

Configurations for keystone v2 and keystone v3 clouds. Config for single tenant scraping.

```
endpoints:
  os-v2-admin:
    kind: openstack
    scope: local
    layouts:
    - hive
    config:
      region_name: RegionOne
      auth:
        username: admin
        password: password
        project_name: admin
        auth_url:  https://keystone-api:5000/v2.0
```

Config for scraping resources from entire cloud.

```
endpoints:
  os-v2-admin:
    kind: openstack
    scope: global
    layouts:
    - hive
    config:
      region_name: RegionOne
      auth:
        username: admin
        password: password
        project_name: admin
        auth_url:  https://keystone-api:5000/v2.0
```

2.4 SaltStack Infrastructures

Configuration for connecting to Salt API.

```
endpoints:
  salt-global:
    kind: salt
    layouts:
    - hive
    config:
      auth_url: http://127.0.0.1:8000
      username: salt-user
      password: password
```

2.5 TerraForm Templates

Configuration for parsing terraform templates.

```
endpoints:
  tf-aws-app:
    kind: terraform
    layouts:
    - hive
    config:
      dir: ~/terraform/two-tier-aws
```

CHAPTER 3

Visualization Layouts

Graph drawing or network diagram is a pictorial representation of the vertices and edges of a graph. This drawing should not be confused with the graph itself, very different layouts can correspond to the same graph. In the abstract, all that matters is which pairs of vertices are connected by edges. In the concrete, however, the arrangement of these vertices and edges within a drawing affects its understandability, usability, fabrication cost, and aesthetics.

The problem gets worse, if the graph changes over time by adding and deleting edges (dynamic graph drawing) and the goal is to preserve the user's mental map.

3.1 Conventions

Graphs are frequently drawn as node-link diagrams in which the vertices are represented as disks, boxes, or textual labels and the edges are represented as line segments, polylines, or curves in the Euclidean plane.

Node-link diagrams can be traced back to the 13th century work of Ramon Llull, who drew diagrams of this type for complete graphs in order to analyze all pairwise combinations among sets of metaphysical concepts.

Alternative conventions to node-link diagrams include:

Adjacency representations such as circle packings, in which vertices are represented by disjoint regions in the plane and edges are represented by adjacencies between regions.

Intersection representations in which vertices are represented by non- disjoint geometric objects and edges are represented by their intersections.

Visibility representations in which vertices are represented by regions in the plane and edges are represented by regions that have an unobstructed line of sight to each other.

Confluent drawings, in which edges are represented as smooth curves within mathematical train tracks.

Fabrics, in which nodes are represented as horizontal lines and edges as vertical lines.

Visualizations of the adjacency matrix of the graph.

3.1.1 Arc Diagram

An arc diagram is a style of graph drawing, in which the vertices of a graph are placed along a line in the Euclidean plane, with edges being drawn as semicircles in one of the two halfplanes bounded by the line, or as smooth curves formed by sequences of semicircles. In some cases, line segments of the line itself are also allowed as edges, as long as they connect only vertices that are consecutive along the line.

The use of the phrase "arc diagram" for this kind of drawings follows the use of a similar type of diagram by Wattenberg (2002) to visualize the repetition patterns in strings, by using arcs to connect pairs of equal substrings. However, this style of graph drawing is much older than its name, dating back to the work of Saaty (1964) and Nicholson (1968), who used arc diagrams to study crossing numbers of graphs. An older but less frequently used name for arc diagrams is *linear embeddings*.

Heer, Bostock & Ogievetsky wrote that arc diagrams "may not convey the overall structure of the graph as effectively as a two-dimensional layout", but that their layout makes it easy to display multivariate data associated with the vertices of the graph.

Sample Visualizations

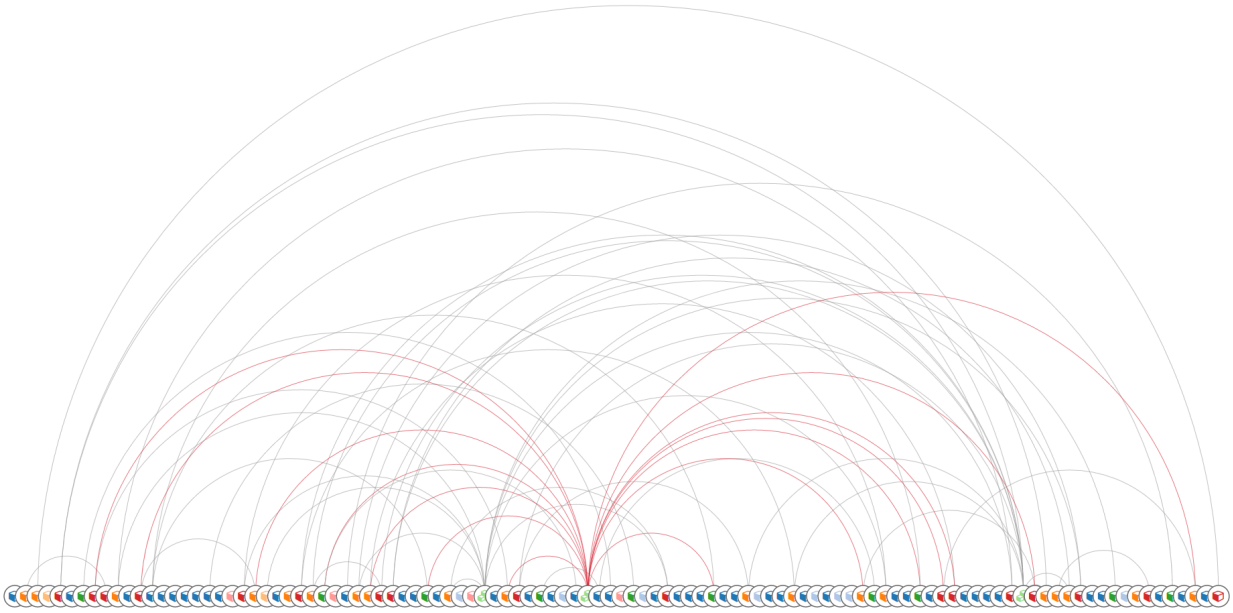

Fig. 3.1: Arc diagram of OpenStack project's resources (cca 100 nodes)

More Information

- https://bl.ocks.org/rpgove/53bb49d6ed762139f33bdaea1f3a9e1c
- https://en.wikipedia.org/wiki/Arc_diagram

3.1.2 Hierarchical Edge Bundling

A compound graph is a frequently encountered type of data set. Relations are given between items, and a hierarchy is defined on the items as well. Hierarchical Edge Bundling is a new method for visualizing such compound graphs. Our

approach is based on visually bundling the adjacency edges, i.e., non-hierarchical edges, together. We realize this as follows. We assume that the hierarchy is shown via a standard tree visualization method. Next, we bend each adjacency edge, modeled as a B-spline curve, toward the polyline defined by the path via the inclusion edges from one node to another. This hierarchical bundling reduces visual clutter and also visualizes implicit adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes. Furthermore, hierarchical edge bundling is a generic method which can be used in conjunction with existing tree visualization techniques.

Sample Visualizations

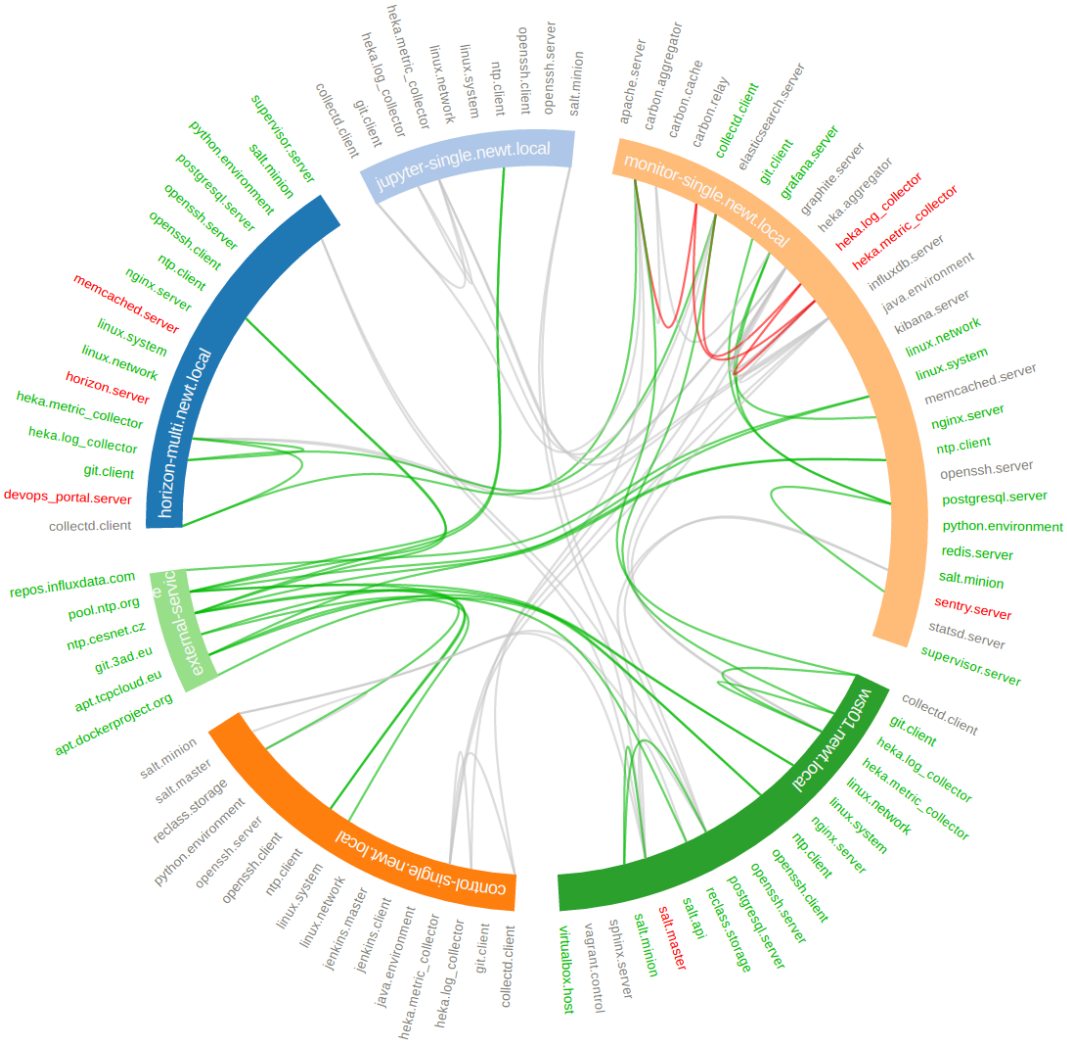

Fig. 3.2: Hierarchical edge bundling of SaltStack services and their relations (cca 100 nodes)

More Information

- http://www.win.tue.nl/vis1/home/dholten/papers/bundles_infovis.pdf
- https://www.win.tue.nl/vis1/home/dholten/papers/forcebundles_eurovis.pdf

3.1.3 Hive Plot

The *hive plot* is a visualization method for drawing networks. Nodes are mapped to and positioned on radially distributed linear axes — this mapping is based on network structural properties. Edges are drawn as curved links. Simple and interpretable.

The purpose of the hive plot is to establish a new baseline for visualization of large networks — a method that is both general and tunable and useful as a starting point in visually exploring network structure.

Sample Visualizations

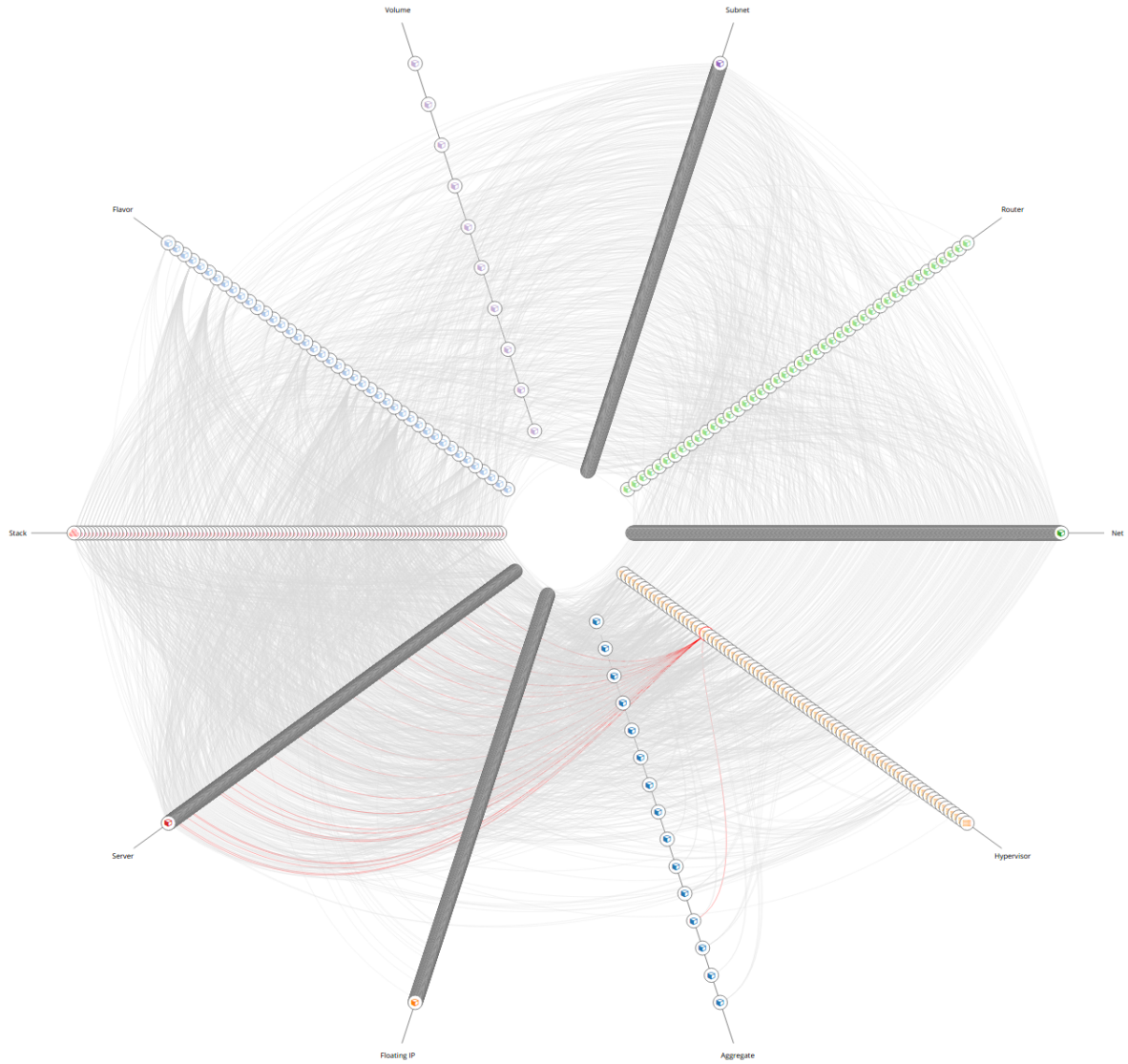

Fig. 3.3: Hive plot of all OpenStack resources (cca 3000 nodes)

More Information

- http://mkweb.bcgsc.ca/linnet/
- https://bost.ocks.org/mike/hive/

3.1.4 Adjacency Matrix

An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected, the adjacency matrix is symmetric. The relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix is studied in spectral graph theory.

The adjacency matrix should be distinguished from the incidence matrix for a graph, a different matrix representation whose elements indicate whether vertex–edge pairs are incident or not, and degree matrix which contains information about the degree of each vertex.

More Information

- https://github.com/micahstubbs/d3-adjacency-matrix-layout
- https://bl.ocks.org/micahstubbs/7f360cc66abfa28b400b96bc75b8984e (Micah Stubbs's adjacency matrix layout)
- https://en.wikipedia.org/wiki/Adjacency_matrix

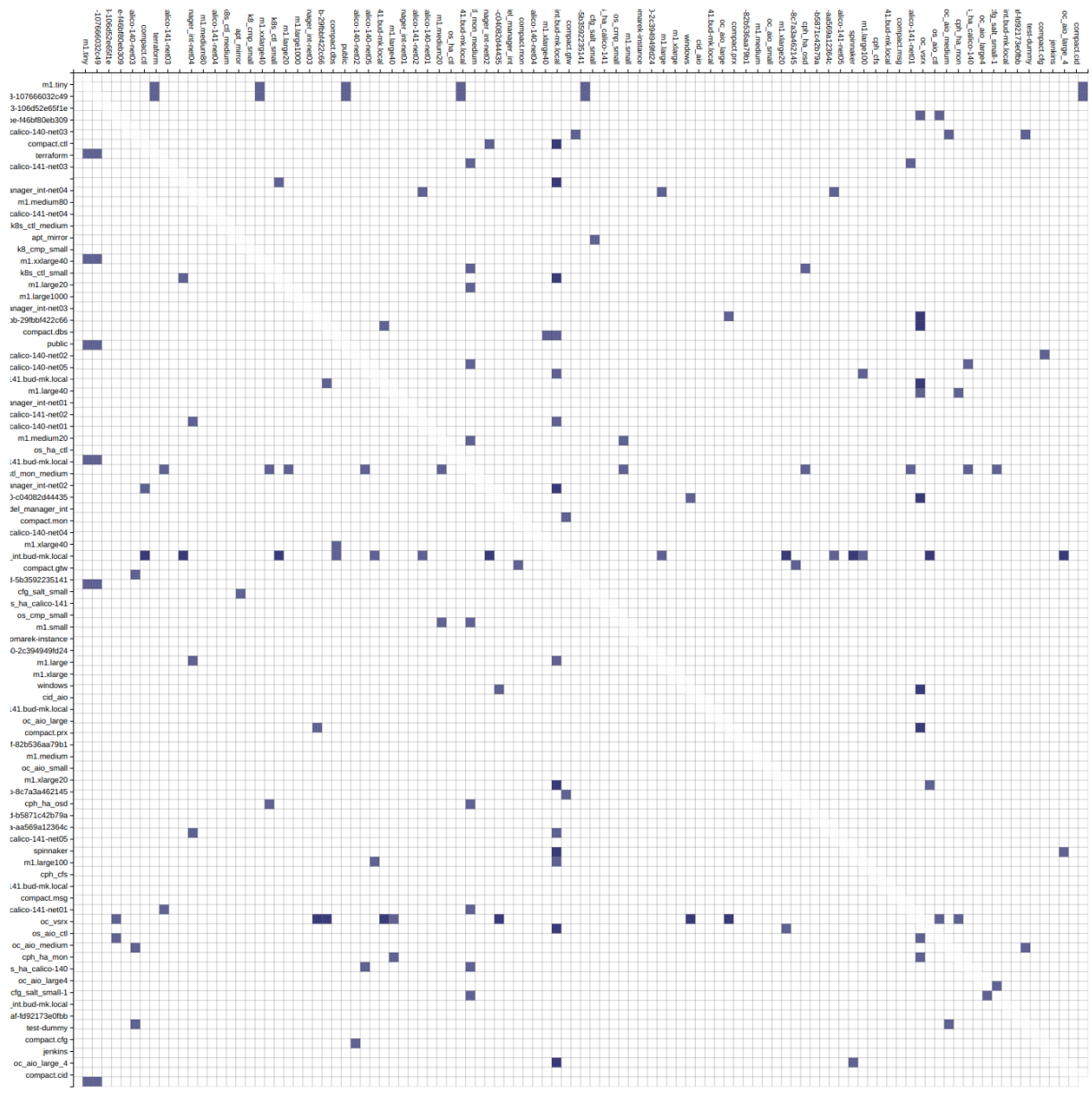

Fig. 3.4: Adjacency matrix of OpenStack project's resources (cca 100 nodes)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search