# The Architect Documentation

***Release 0.2.0***

**The Architect Team**

**Dec 04, 2018**

# Contents

CHAPTER 1

# The Architect System

## 1.1 Architect Introduction

The aim of this project is to provide unified service modeling, management and visualization platform agnostic of delivery or orchestration method. It creates virtual representations of any software services or physical resources and allows control over crucial steps in their life cycle. Both reductionist and holistic approaches are used to descibe target systems. The project name comes from Architect program in Matrix movie series:

> In the Matrix the Architect is a highly specialized, humorless program of the machine world as well as the creator of the Matrix. As the chief administrator of the system, he is possibly a collective manifestation, or at the very least a virtual representation of the entire Machine mainframe.

The The Architect project was started as part of my thesis "Visualization of cloud performace metrics". Now we explore the possible implications of combining the relational models of infrastructures with quantitative data that relates to it. This the implementation of holistic approach to the IT system modeling. You combine the capabilities of the brain (inventory), muscles (manager) and senses (monitor) to create the full body of IT system. This vague analogy, but you seldom see all the parts of infrastrucure working together as one, the source of truth providing the vital information to the orchestration engines and configuring the monitoring to reflect the actual state. Then you can start implementing your policy engines and machine learching techiques to improve the state of your initial models. This is not possible to achieve withnout proper decomposition your system to individual pieces but also you need to put it back together and look at it as whole.

### 1.1.1 The Project History

Academic research in particular fields has been undergoing since 2013. We have published series of research papers covering in detail specificic areas of capabilities that became part of Architect project.

In 2014 we got published *Security information and event management in the cloud computing infrastructure* and presented at CINTI 2014 - 15th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings.

In 2015 we got published *Opensource automation in cloud computing* at Lecture Notes in Electrical Engineering and *Network visualization survey* at Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial

Intelligence and Lecture Notes in Bioinformatics).

Also *High level models for IaaS cloud architectures* published at Studies in Computational Intelligence and *Measurement of cloud computing services availability* published in Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST made in year 2015.

In 2017 we got published *Hybrid system orchestration with TOSCA and salt* in Journal of Engineering and Applied Sciences and *VNF orchestration and modeling with ETSI MANO compliant frameworks* which got published at Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

We tried to identify possible options to model cloud architectures, ontologies, definition hiearchies. The good, the bad and the ugly of metadata. Then we focused on ways to manage orchestration processes, not only for compute servers but also for virtual network resources. Some works focuced on measuring cloud metrics and evaluating log event data important to undestand the types of data the systems emit and how to normalise these to consitent domains. And last type of works, some of them unpublished, were concerned with the visualization layout methods, transformation techniques for relational and quatitative data. The individual reseach papers were focusec on gaining expertise in given domain. The Architect project wraps the outcome of individual research into consistent holistic framework for modeling complete IT infrastructures.

### 1.1.2 PhD Thesis Abstract

This thesis provides implementation of platform for holistic system modeling. It tries to define the main components of system governance, that are required for autonomous long-term opertations and interactions. These components are the brain, source of truth that provides models to the muscles and senses. The orchestration platforms or cloud service providers are the virtual muscles. The senses are the tools that create and process the metrics and event data. Usually sences are backed by multiple levels of human powered support teams. The proposed platform allows combination of brain, muscles and senses to create entity which can displayes in different perspectives by well tailored visualizations that reflect the actual life-cycle state of the governed infrastructures.

The main part of work presents implementation of Architect project, that has been developed as proof-of-concept implementation of holistic governance system capable of modern user and systems interactions. It can display service relations and gathered metrics and give advanced insight to important Cloud computing performance qualities. Modern systems rely closely on cloud and virtualisation architectures. To optimize utilization of resources, applications and infrastructure must not be controlled separately.

We propose a united platform that can discribe major orchestration engines and monitoring solutions in common schema, create metadata inventories that can provide proper context to any of these services. On top of this infrastructure models, you have powerful interfactive visualizations that can be used to display properties of interest. On other side the models of actual infrastructure states are good entrypoint for further machine assisted analysis and learning techniques.

#### Keywords

Cloud Computing, Metadata, Virtualization, Metering, Monitoring, SOA, Data Transformation, Data Visualization, Chart, Time-series, Event, Holism, Service Science, System Science

#### Resources

- Documentation: https://architect-api.readthedocs.io/en/latest/
- API source: https://github.com/cznewt/architect-api
- Client source: https://github.com/cznewt/architect-client
- Research papers: https://www.scopus.com/authid/detail.uri?authorId=56501819000

## 1.2 Architect Components

Following figure shows high-level achitecture of Architect system.

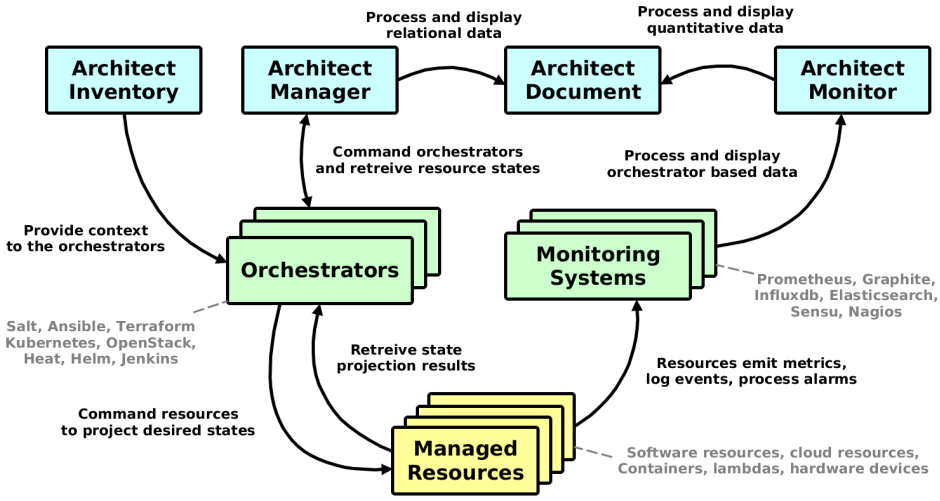



Fig. 1: High-level achitecture of Architect system

The Architect project consists of 4 core compontents. A quick summary of properties, capabilities and integrations for each component.

### 1.2.1 Inventory Component

Inventory is the Architect's metadata engine. It encapsulates and unifies data from various metadata sources to provide inventory metadata for various orchestration services. Basically serves as metadata proxy with clients. It works best integrated with http://salt-formulas.readthedocs.io/.

Currently supported metadata backends are:

- salt-formulas
- reclass (python3 version)

The currently supported customers of metadata provided by Inventory using `architect-api` client library are:

- SaltStack
- Ansible
- Puppet
- Chef

Following orchestrators have direct support for injecting context metadata:

- Heat

### 1.2.2 Manager Component

Manager is the Architect's orchestration engine. The aim of this module is to enforce infrastructure topologies models and acquire live infrastructure topology data from any resource provider for further relational and quantitative analysis

and visualisations.

The pull approach for querying endpoint APIs is supported at the moment, the processing push from target services is supported for SaltStack events.

Currently supported resource providers are:

- Kubernetes clusters
- OpenStack clouds
- Heat templates
- Amazon web services
- SaltStack infrastructures
- Terraform templates
- Jenkins pipelines

### 1.2.3 Monitor Component

Monitor is the Architect's monitoring engine. It can connect to multiple data endpoints and subject them for further analysis. We can define queries for quantitative data or time-series in Document component.

Currently supported monitoring services are:

- Graphite
- ElasticSearch
- Prometheus
- InfluxDB

### 1.2.4 Document Component

Document component is responsible for analysis and visualization of infrastructure resources in form of directed graph. We can perform several transformation functions on this graph data. The other part is analysis of quantitative data provided by monitoring solutions and corellating it to the relational structures provided by Manager component.

Currently supported relational visualization layouts:

- Adjacency matrix
- Arc diagram
- Force-directed graph
- Hierarchical edge bundling
- Hive plot
- Circle packing
- Node-link tree (Reingold-Tilford tidy trees, dendrograms)
- Partition layout (sunburst, icicle diagrams, treemaps)
- Sankey diagram

Currently supported quatitative visualization layouts:

- Line chart

- Bar chart, stacked bar chart
- Horizon chart
- Donut chart, pie chart

## 1.3 Architect Services Installation

Following steps show how to deploy various components of the Architect service and connections to external services. It covers the basic development deployment.

### 1.3.1 Service `architect-api` Installation

The core service responsible for handling HTTP API requests and providing simple UI based on Material design. Release version of architect-api is currently available on Pypi, to install it, simply execute:

```
pip install architect-api
```

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

```
virtualenv -p python3 venv
source venv/bin/activate

git clone git@github.com:cznewt/architect-api.git

cd architect-api

python setup.py install
```

Or you you can install service by pip `architect-api` package.

```
virtualenv venv
source venv/bin/activate

pip install architect-api -e
```

#### Initial Setup for UI

Architect-api uses the `npm` to install its JavaScript dependencies, which are collected by `django-npm` static files collector. You can install all static nodesj libraries by following commands. More about installing Node.js and NPM can be found at https://www.npmjs.com/get-npm.

```
apt install npm

cd architect-api

npm install
```

Architect uses the Bootstrap 4 library wich uses SASS 3.5 style preprocessing. No python SASS interpreter does it well so we need to get the ruby's gems this time. The static file compress utility uses this ruby binary to perform the processing of SASS styles. You can install the SASS compiler by following commands. More about installing SASS can be found at http://sass-lang.com/install.

```
apt install gem ruby-dev

sudo gem install sass --no-user-install

sass --version
```

The `saas --version` command should return Sass 3.5 or higher.

Now you can collect all your static assets, run following command in architect base dir and sourced.

```
$ python manage.py collectstatic --noinput

X static files copied to '/python-apps/architect/static', Y unmodified.
```

Now you can compress and compile your static assets, in architect base dir and sourced run following command.

```
$ python manage.py compress

Found 'compress' tags in:
    /python-apps/architect/architect/templates/_head.html
    /python-apps/architect/architect/templates/_body.html
```

### Initial Setup for Database

You must synchronise your database content with the current migration scheme, command will create entire schema and apply all the migrations if run for the first time. In architect base dir and sourced run following command.

```
python manage.py migrate
```

You need also setup your user credentials if creating a new deployment.

```
python manage.py createsuperuser
```

You can install sample metadata fixtures by following command.

```
$ python manage.py loaddata sample_saltstack

Installed 614 object(s) from 2 fixture(s)
```

You must set database configuration by settings in architect-api configuration file. Example PostgreSQL settings in architect-api configuration file.

```
databases:
  default:
    ENGINE: django.db.backends.postgresql_psycopg2
    NAME: architect
    USER: architect
    PASSWORD: password
    HOST: 127.0.0.1
    PORT: 5432
```

The similar applies for the cache backend, which can be changed to the Memcached backend, for example:

```
caches:
  default:
```

(continues on next page)

(continued from previous page)

```
    BACKEND: django.core.cache.backends.memcached.MemcachedCache
    LOCATION: 127.0.0.1:11211
```

### Main Configuration File

You provide one YAML configuration file for all settings. The default location is `/etc/architect/api.yml`.

You can setup basic configuration of database and cache also you can provide defaults for your initial inventories, managers and monitors.

You can override the default location of the configuration file by setting the `ARCHITECT_CONFIG_FILE` environmental variable to your custom location.

The configuration file currently supports following options:

```
databases:
  default:
    ENGINE: django.db.backends.postgresql_psycopg2
    ...
caches:
  default:
    BACKEND: django.core.cache.backends.memcached.MemcachedCache
    ...
monitor:
  monitor01:
    name: Dashboard 01
    ...
manager:
  manager01:
    engine: salt
    ...
inventory:
  inventory01:
    engine: reclass
    ...
```

The `databases` and `caches` keys are used in the application settings. But the `monitor`, `manager` and `inventory` configuration settings need to be sychronised to database by management commands in architect base dir and sourced.

```
$ python manage.py sync_inventories

Inventory "inventory01" resource updated
...

$ python manage.py sync_managers

Manager "manager01" resource updated
...

$ python manage.py sync_monitors

Monitor "monitor01" resource updated
...
```

You can run the configuration multiple times and update existing resources. The actual resources used are stored in

the database and can be changed at the architect's admin app available at http://127.0.0.1:8181/admin/ after you start the development server.

Look at the the documentation pages for individual inventory, manager or monitor configuration options and installation problems.

#### Running Development Server

To start development server, in architect base dir and sourced run following command.

```
$ python manage.py runserver 0.0.0.0:8181

Performing system checks...

System check identified no issues (0 silenced).
January 27, 2018 - 13:12:47
Django version 2.0.1, using settings 'architect.settings'
Starting development server at http://0.0.0.0:8181/
Quit the server with CONTROL-C.
```

### 1.3.2 Service `architect-worker` Installation

The architect relies on standalone workers to perform the tasks asynchronously. For the development environment, you can just simply install redis server to serve as message bus by following command.

```
apt install redis server
```

Now you can start running your architect worker instances. The redis is hardcoded and celery can be replaced by airflow, this is up to discussion.

#### Running development worker

To start development worker, in architect base dir and sourced run following command.

```
$ celery -A architect worker -l info

 -------------- celery@wst01 v4.1.0 (latentcall)
---- **** -----
--- * ***  * -- Linux-4.10.0-42
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app:         architect:0x7ff566a38e80
- ** ---------- .> transport:   redis://localhost:6379//
- ** ---------- .> results:     redis://localhost:6379/
- *** --- * --- .> concurrency: 4 (prefork)
-- ******* ---- .> task events: OFF
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery

[tasks]
  . architect.celery.debug_task
  . get_manager_status_task
```

(continues on next page)

(continued from previous page)

```
[2018-01-27 13:15:55,852: INFO/MainProcess] Connected to redis://localhost:6379//
[2018-01-27 13:15:55,860: INFO/MainProcess] mingle: searching for neighbors
[2018-01-27 13:15:56,880: INFO/MainProcess] mingle: all alone
[2018-01-27 13:15:56,892: INFO/MainProcess] celery@<your-node-hostname> ready.
```

You should see `celery@<your-node-hostname> ready` in the output of the command run. If not, check if redis service `systemctl status redis-server` is running. You need at least one instance of worker running.

### 1.3.3 Service `architect-client` Installation

Following steps show how to deploy and configure Architect Client. You need to install client on configuration management servers to integrate the inventory service.

```
pip install architect-client
```

Create configuration file `/etc/architect/client.yml` for client.

```
project: project-name
host: architect-api
port: 8181
username: salt
password: password
```

### 1.3.4 Complete Installation Scripts

#### Architect API Dependencies

You can install requried services.

```
#!/bin/bash -ex

export DEBIAN_FRONTEND=noninteractive
export LC_ALL=en_US.utf8

printf "Update Ubuntu ..."
sudo apt-get update -y

printf "Install servers ..."
apt-get -y install memcached redis-server postgresql-9.5

printf "Setup Postgresql ..."
sudo -u postgres psql -c "CREATE DATABASE architect"
sudo -u postgres psql -c "CREATE USER architect WITH PASSWORD 'password'"
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE architect TO architect"
```

Or use local `docker-compose.yml` to start the same services.

#### Architect API Core

Install full development environment.

```
#!/bin/bash -ex

export DEBIAN_FRONTEND=noninteractive
export LC_ALL=en_US.utf8

printf "Update Ubuntu ..."
sudo apt-get update -y

printf "Installing Python 3 dependencies..."
apt-get -y install python-virtualenv python3-dev python3-pip libxml2-dev libxslt1-dev␣
↪libffi-dev graphviz libpq-dev libssl-dev

printf "Upgrading pip"
pip3 install --upgrade pip

mkdir -p /etc/architect
cat << EOF > /etc/architect/api.yml
databases:
  default:
    ENGINE: django.db.backends.postgresql_psycopg2
    NAME: architect
    USER: architect
    PASSWORD: password
    HOST: 127.0.0.1
    PORT: 5432
caches:
  default:
    BACKEND: django.core.cache.backends.memcached.MemcachedCache
    LOCATION: 127.0.0.1:11211
inventory:
  sample-cluster:
    engine: hier-cluster
    service_class_dir:
    system_class_dir:
    cluster_class_dir:
    class_dir: /srv/architect/mcp/classes
    formula_dir: /srv/salt-formulas/formulas
  sample-deploy:
    engine: hier-deploy
    class_dir: /srv/architect/mcp/classes
    node_dir: /srv/architect/mcp/nodes/sample-deploy
EOF

git clone https://github.com/cznewt/architect-api.git /opt/architect
cd /opt/architect

printf "Installing architect-api code"
virtualenv -p python3 venv
. venv/bin/activate
pip install -r requirements/base.txt
pip install psycopg2-binary
pip install git+https://github.com/salt-formulas/reclass.git@python3

printf "Installing static files tools"
apt-get -y install npm rubygems ruby-dev

sudo gem install sass --no-user-install
```

(continues on next page)

(continued from previous page)

```
npm install

python manage.py collectstatic --noinput
python manage.py compress
python manage.py migrate

# Now you can run following 2 services

# python manage.py runserver 0.0.0.0:8181

# celery -A architect worker -l info
```

### Repository Image Builders

```
#!/bin/bash -ex

export DEBIAN_FRONTEND=noninteractive
export LC_ALL=en_US.utf8

printf "Update Ubuntu ..."
sudo apt-get update -y

printf "Install Raspberry Pi build dependencies ..."
apt-get install -y debootstrap debian-archive-keyring qemu-user-static binfmt-support␣
↪dosfstools bmap-tools whois bc crossbuild-essential-armhf

printf "Install BeagleBone build dependencies ..."
apt-get install -y m4 bmap-tools dosfstools rsync git-core kpartx wget parted pv
```

### Sample Hierarchical Inventory

```
#!/bin/bash -ex

mkdir -p /srv/architect/mcp/classes/deployment
mkdir -p /srv/architect/mcp/classes/service
mkdir -p /srv/architect/mcp/classes/cluster/sample/infra
mkdir -p /srv/architect/mcp/nodes/sample-deploy

if [ ! -d "/srv/architect/mcp/classes/system" ]; then
  git clone https://github.com/Mirantis/reclass-system-salt-model /srv/architect/mcp/
↪classes/system
fi;

if [ ! -d "/srv/salt-formulas" ]; then
  git clone https://github.com/salt-formulas/salt-formulas.git --recursive /srv/salt-
↪formulas
fi;

for i in /srv/salt-formulas/formulas/* ; do
  if [ -d "$i" ]; then
    service=$(basename "$i")
    formula="${service/-/_}"
```

(continues on next page)

(continued from previous page)

```
    if [ -d "/srv/salt-formulas/formulas/$service/metadata/service" ]; then
      if [ ! -L "/srv/architect/mcp/classes/service/$formula" ]; then
        ln -s "/srv/salt-formulas/formulas/$service/metadata/service" "/srv/architect/
↪mcp/classes/service/$formula"
      fi;
    fi;
  fi;
done

cat << EOF > /srv/architect/mcp/classes/cluster/sample/infra/config.yml
classes:
- service.git.client
- system.linux.system.single
- system.linux.system.repo.mcp.salt
- system.salt.master.pkg
parameters:
  _param:
    salt_master_host: 127.0.0.1
    salt_master_environment_name: dev
    salt_master_environment_repository: https://github.com/salt-formulas
    salt_master_environment_revision: master
    salt_minion_ca_authority: salt_master_ca
  salt:
    master:
      user:
        salt:
          permissions:
          - '.*'
          - '@local'
          - '@wheel'   # to allow access to all wheel modules
          - '@runner'  # to allow access to all runner modules
          - '@jobs'    # to allow access to the jobs runner and/or wheel modules
      engine:
        architect:
          engine: architect
          project: sample-deploy
          host: 127.0.0.1
          port: 8181
          username: architect
          password: password
EOF

cat << EOF > /srv/architect/mcp/classes/deployment/sample-deploy.yml
parameters:
  _param:
    cluster_name: sample-deploy
    cluster_domain: sample.deploy
EOF

cat << EOF > /srv/architect/mcp/nodes/sample-deploy/cfg01.sample.deploy.yml
classes:
- cluster.sample.infra.config
- deployment.sample-deploy
parameters:
  _param:
    linux_system_codename: xenial
  linux:
```

(continues on next page)

(continued from previous page)

```
    system:
      name: cfg01
      domain: sample.deploy
EOF
```

CHAPTER 2

# Architect Inventory

Each manager endpoint expects different configuration. Following samples show the required parameters to setup individual inventory backends.

## 2.1 Architect Inventory Backends

### 2.1.1 Reclass Inventory

Following configuration points to the reclass inventory storage on local filesystem.

```
inventory:
  reclass-inventory:
    storage_type: yaml_fs
    class_dir: /srv/salt/reclass/classes
    node_dir: /srv/salt/reclass-nodes/{{ inventory-name }}
```

#### References

- http://reclass.pantsfullofunix.net/
- https://github.com/salt-formulas/reclass

### 2.1.2 Hierarchical Inventory

#### Hier-cluster Inventory

Following configuration points to the hiearchical-cluster inventory storaged on local filesystem.

```
inventory:
  hier-cluster-inventory:
    engine: hier-cluster
    service_class_dir:
    system_class_dir:
    cluster_class_dir:
    class_dir: /srv/salt/reclass/classes
    formula_dir: /srv/salt/base
    form:
      test:
        name: Generate new cluster class
        layout: page
        templates:
        - file: cluster/{{ cluster_name }}.yml
          content: |
            parameters:
              meta: {{ cluster_name }}
              {%- if cluster_domain %}
              domain: {{ cluster_name }}
              {%- endif %}
        steps:
        - name: form-single
          layout: inline
          fields:
          - name: cluster_name
            label: Cluster name
            value_type: string
            help_text: Name of the cluster, used as cluster/CLUSTER_NAME.yml in␣
↪directory structure.
            initial: deployment_name
            style: col-md-6
          - name: cluster_domain
            label: Cluster domain
            value_type: boolean
            help_text: Is cluster appended?
            initial: deploy-name.local
            width: half
            style: col-md-6
```

### Hier-deploy Inventory

Following configuration points to the hiearchical-deploy inventory storaged on local filesystem.

```
inventory:
  hier-deploy-inventory:
    engine: hier-deploy
    class_dir: /srv/salt/reclass/classes
    node_dir: /srv/salt/reclass/nodes
```

### References

- http://salt-formulas.readthedocs.io/

## 2.2 Architect Inventory Consumers

### 2.2.1 SaltStack Consumer

#### `http_architect` pillar backend

To enable Salt Master inventory, you need to install `http_architect` Pillar and Top modules and add following to the Salt Master configuration files. To support the grains.

```
http_architect: &http_architect
  project: local-salt
  host: architect.service.host
  port: 8181

ext_pillar:
  - http_architect: *http_architect

master_tops:
  http_architect: *http_architect
```

#### Native pillar backend

To setup architect as Salt master Pillar source, set following configuration to your Salt master at `/etc/salt/master.d/_master.conf` file.

```
ext_pillar:
  - cmd_yaml: 'architect-salt-pillar %s'
```

To setup architect as Salt master Tops source, set following configuration to your Salt master at `/etc/salt/master.d/_master.conf` file.

```
master_tops:
   ext_nodes: architect-salt-top
```

You can test the SaltStack Pillar by calling command:

```
$ architect-salt-pillar {{ minion-id }}
```

#### References

- https://docs.saltstack.com/en/latest/ref/tops/all/salt.tops.ext_nodes.html
- https://docs.saltstack.com/en/latest/ref/pillar/all/salt.pillar.cmd_yaml.html#module-salt.pillar.cmd_yaml

### 2.2.2 Ansible Consumer

To setup architect as Ansible dynamic inventory source, set following configuration to your Ansible control node.

```
$ ansible -i architect-ansible-inventory
```

You can test the ansible inventory by calling command:

```
$ architect-ansible-inventory --list
```

#### References

- http://docs.ansible.com/ansible/latest/dev_guide/developing_inventory.html

### 2.2.3 Puppet Consumer

To tell Puppet Server to use an ENC, you need to set two settings: `node_terminus` has to be set to "exec", and `external_nodes` must have the path to the executable.

```
[master]
  node_terminus = exec
  external_nodes = /usr/local/bin/architect-puppet-classifier
```

#### References

- https://puppet.com/docs/puppet/5.3/nodes_external.html

### 2.2.4 Chef Consumer

We can use `-j` parameter of `chef-client` command, It's the path to a file that contains JSON data used to setup the client run. We pass

```
$ architect-chef-data {{ node_name }} {{ file_name }}.json
$ chef-client -j {{ file_name }}.json --environment _default
```

#### References

- https://docs.chef.io/ctl_chef_client.html

CHAPTER 3

# Architect Manager

## 3.1 Configuration Management

### 3.1.1 SaltStack Infrastructures

Configuration for connecting to Salt API endpoint.

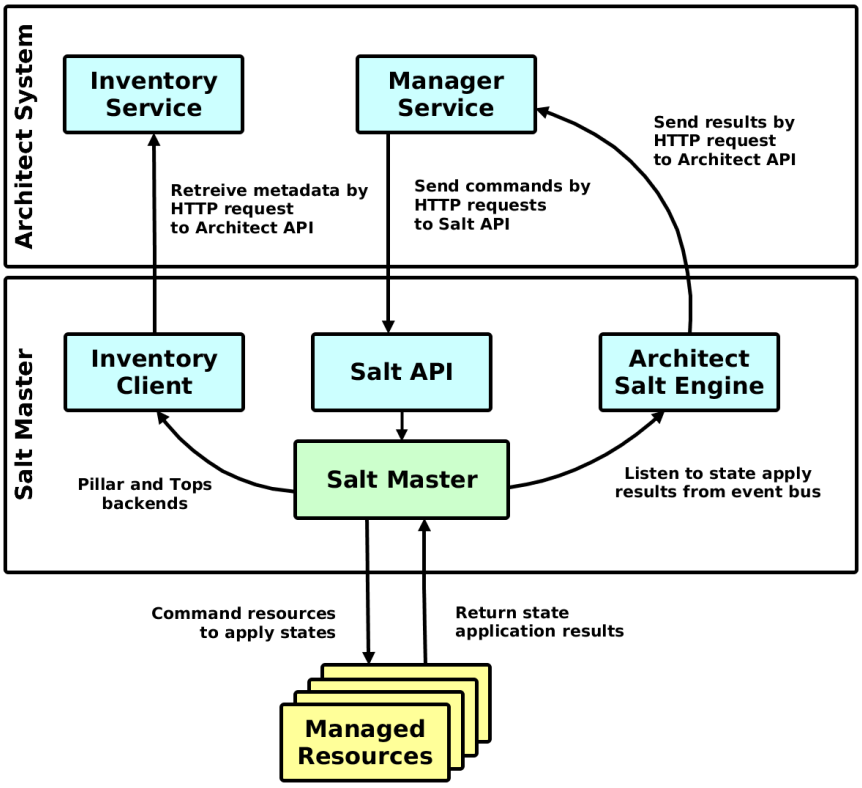
```
monitor:
  salt-manager:
    auth_url: http://{{ salt-api }}:8000
    username: {{ user-name }}
    password: {{ user-password }}
```

Following figure shows how SaltStack integrates with Architect Inventory and Manager. Please note that you can use Inventory integration independently of the Manager integration.

The metadata schema for SaltStack manager:

```
query:
  salt_complete_graph:
    name: All Resources
    layout: graph
  salt_minion_service_graph:
    name: Salt Minion Services
    layout: graph
    filter_node_types:
    - salt_master
    - salt_minion
    - salt_service
  salt_minions_tree:
    name: Simple Minions
    layout: hierarchy
    hierarchy_layers:
      0:
```

(continues on next page)




(continued from previous page)

```
      name: Salt Master
      kind:
    1:
      kind: salt_minion
salt_minion_services_tree:
  name: Minion Services
  layout: hierarchy
  hierarchy_layers:
    0:
      name: Salt master
      kind:
    1:
      kind: salt_minion
    2:
      target: runs_on_minion
      kind: salt_service
salt_minion_lowstates_tree:
  name: Minion States
  layout: hierarchy
  hierarchy_layers:
    0:
      name: Salt master
      kind:
    1:
      kind: salt_minion
    2:
      target: runs_on_minion
      kind: salt_service
```

(continues on next page)

(continued from previous page)

```
      3:
        target: state_of_service
        kind: salt_lowstate
relation:
  controlled_by_master:
    relation:
      default: master
  applied_on_minion:
    relation:
      default: minion
  runs_on_minion:
    relation:
      default: minion
  applied_lowstate:
    relation:
      default: lowstate
  state_of_service:
    relation:
      default: service
  action_by_user:
    relation:
      default: user
  requires_service:
    relation:
      default: require
default_resource: salt_minion
resource:
  salt_master:
    client: ''
    icon: fa:server
    name: Master
    resource: master
    workflow:
      generate_key:
        name: Generate key

  salt_minion:
    client: ''
    icon: fa:server
    name: Minion
    resource: minion
    workflow:
      run_module:
        name: Run module
    model:
      master:
        type: relationship_to
        model: controlled_by_master
        target: salt_master
  salt_lowstate:
    client: ''
    icon: fa:cube
    name: Lowstate
    resource: lowstate
    model:
      service:
        type: relationship_to
```

(continues on next page)

(continued from previous page)

```
        model: state_of_service
        target: salt_service
  salt_job:
    client: ''
    icon: fa:clock-o
    name: Job
    resource: job
    model:
      user:
        type: relationship_to
        model: action_by_user
        target: salt_user
      minion:
        type: relationship_to
        model: applied_on_minion
        target: salt_minion
      lowstate:
        type: relationship_to
        model: applied_lowstate
        target: salt_lowstate
  salt_service:
    client: ''
    icon: fa:podcast
    name: Service
    resource: service
    model:
      minion:
        type: relationship_to
        model: runs_on_minion
        target: salt_minion
      require:
        type: relationship_to
        model: requires_service
        target: salt_service
  salt_user:
    client: ''
    icon: fa:user
    name: User
    resource: user
```

### Salt Master Integration

You can control salt master infrastructure and get the status of managed hosts and resources. The Salt engine `architect` relays the state outputs of individual state runs and `architect` runners and modules provide the capabilities to interface with salt and architect functions. The Salt Master is managed through it's HTTP API service.

```
http_architect: &http_architect
  project: newt.work
  host: 127.0.0.1
  port: 8181
```

## 3.2 Cloud Resource Management

### 3.2.1 Amazon Web Services

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

```
region: us-west-2
aws_access_key_id: {{ access_key_id }}
aws_secret_access_key: {{ secret_access_key }}
```

The metadata schema for AWS manager:

```
query:
  aws_complete_graph:
    name: Complete Account
    layout: graph
relation:
  in_ec2_vpc:
    relation:
      default: vpc
  in_ec2_subnet:
    relation:
      default: subnet
  using_ec2_key_pair:
    relation:
      default: key_pair
default_resource: ec2_instance
resource:
  ec2_elastic_ip:
    client: ec2
    icon: fa:cube
    name: Elastic IP
    resource: AWS::EC2::EIP
  ec2_elastic_ip_association:
    client: ec2
    icon: fa:cube
    name: Elastic IP Association
    resource: AWS::EC2::EIPAssociation
  ec2_image:
    client: ec2
    icon: fa:server
    name: Image
    resource: AWS::EC2::Image
  ec2_instance:
    client: ec2
    icon: fa:server
    name: Instance
    resource: AWS::EC2::Instance
    model:
      vpc:
        type: relationship_to
        model: in_ec2_vpc
        target: ec2_vpc
      subnet:
        type: relationship_to
        model: in_ec2_subnet
        target: ec2_subnet
```

(continues on next page)

(continued from previous page)

```
      key_pair:
        type: relationship_to
        model: using_ec2_key_pair
        target: ec2_key_pair
  ec2_internet_gateway:
    client: ec2
    icon: fa:cube
    name: Internet Gateway
    resource: AWS::EC2::InternetGateway
  ec2_key_pair:
    client: ec2
    icon: fa:key
    name: Key Pair
    resource: AWS::EC2::KeyPair
  ec2_route_table:
    client: ec2
    icon: fa:cube
    name: Route Table
    resource: AWS::EC2::RouteTable
  ec2_security_group:
    client: ec2
    icon: fa:cubes
    name: Security Group
    resource: AWS::EC2::SecurityGroup
  ec2_subnet:
    client: ec2
    icon: fa:cube
    name: Subnet
    resource: AWS::EC2::Subnet
  ec2_vpc:
    client: ec2
    icon: fa:cubes
    name: VPC
    resource: AWS::EC2::VPC
  ec2_vpc_gateway_attachment:
    client: ec2
    icon: fa:cube
    name: VPC Gateway Attachment
    resource: AWS::EC2::VPCGatewayAttachment
  s3_bucket:
    client: s3
    icon: fa:hdd-o
    name: Bucket
    resource: AWS::S3::Bucket
```

### 3.2.2 OpenStack Cloud Resources

Configuration for keystone `v2.0` and keystone `v3` clouds. Configuration sample for single tenant access.

```
scope: local
region_name: RegionOne
auth:
  username: {{ user-name }}
  password: {{ user-password }}
  project_name: {{ project-name }}
```

(continues on next page)

(continued from previous page)

```
  auth_url: https://{{ keystone-api }}:5000/v2.0
```

Config for managing resources of entire cloud, including hypervisors, tenants, etc in given region.

```
scope: global
region_name: RegionOne
auth:
  username: {{ admin-name }}
  password: {{ admin-password }}
  project_name: admin
  auth_url: https://{{ keystone-api }}:5000/v2.0
```

The metadata schema for OpenStack manager:

```
query:
  os_complete_graph:
    name: Complete Cloud
    layout: graph
  os_clear_project_graph:
    name: Project Resources
    layout: graph
    filter_node_types:
    - os_server
    - os_key_pair
    - os_flavor
    - os_network
    - os_subnet
    - os_floating_ip
    - os_port
    - os_router
    filter_lone_nodes:
    - os_key_pair
    - os_flavor
  os_network_topology_graph:
    name: Network Topology
    layout: graph
    filter_node_types:
    - os_server
    - os_network
    - os_subnet
    - os_port
    - os_floating_ip
    - os_router
relation:
  in_os_project:
    relation:
      default: project
  in_os_stack:
    relation:
      default: stack
  in_os_network:
    relation:
      default: network
  use_os_port:
    relation:
      default: port
      os_server: network
```

(continues on next page)

(continued from previous page)

```
  use_os_image:
    relation:
      os_server: image
  use_os_flavor:
    relation:
      os_server: flavor
  use_os_key_pair:
    relation:
      default: key_pair
  on_os_hypervisor:
    relation:
      os_server: hypervisor
      os_port: hypervisor
  in_os_aggregate:
    relation:
      os_hypervisor: aggregate
default_resource: os_server
resource:
  os_aggregate:
    resource: OS::Nova::Aggregate
    client: nova
    name: Aggregate
    icon: fa:cube
  os_flavor:
    resource: OS::Nova::Flavor
    client: nova
    name: Flavor
    icon: fa:cube
  os_floating_ip:
    resource: OS::Neutron::FloatingIP
    client: neutron
    name: Floating IP
    icon: fa:cube
    model:
      stack:
        type: relationship_to
        model: in_os_stack
        target: os_stack
      project:
        type: relationship_to
        model: in_os_project
        target: os_project
      port:
        type: relationship_to
        model: use_os_port
        target: os_port
  os_floating_ip_association:
    resource: OS::Neutron::FloatingIPAssociation
    client: neutron
    name: Floating IP Association
    icon: fa:cube
  os_group:
    resource: OS::Keystone::Group
    client: keystone
    name: Group
    icon: fa:cube
  os_hypervisor:
```

(continues on next page)

(continued from previous page)

```
    resource: OS::Nova::Hypervisor
    client: nova
    name: Hypervisor
    icon: fa:server
    model:
      aggregate:
        type: relationship_to
        model: in_os_aggregate
        target: os_aggregate
  os_image:
    resource: Glance::Image
    client: glance
    name: Image
    icon: fa:cube
  os_key_pair:
    resource: OS::Nova::KeyPair
    client: nova
    name: Key Pair
    icon: fa:key
    model:
      stack:
        type: relationship_to
        model: in_os_stack
        target: os_stack
  os_network:
    resource: OS::Neutron::Net
    client: neutron
    name: Network
    icon: fa:share-alt
    model:
      stack:
        type: relationship_to
        model: in_os_stack
        target: os_stack
      project:
        type: relationship_to
        model: in_os_project
        target: os_project
      subnets:
        type: relationship_from
        model: in_os_network
        target: os_subnet
  os_port:
    resource: OS::Neutron::Port
    client: neutron
    name: Port
    icon: fa:cube
    model:
      stack:
        type: relationship_to
        model: in_os_stack
        target: os_stack
      project:
        type: relationship_to
        model: in_os_project
        target: os_project
      hypervisor:
```

(continues on next page)

(continued from previous page)

```
      type: relationship_to
      model: on_os_hypervisor
      target: os_hypervisor
    network:
      type: relationship_to
      model: in_os_network
      target: os_network
os_project:
  resource: OS::Keystone::Tenant
  client: keystone
  name: Project
  icon: fa:cube
os_resource_type:
  resource: OS::Heat::ResourceType
  client: heat
  name: Resource Type
  icon: fa:cube
os_router:
  resource: OS::Neutron::Router
  client: neutron
  name: Router
  icon: fa:arrows-alt
  model:
    stack:
      type: relationship_to
      model: in_os_stack
      target: os_stack
    project:
      type: relationship_to
      model: in_os_project
      target: os_project
    port:
      type: relationship_to
      model: use_os_port
      target: os_port
os_server:
  resource: OS::Nova::Server
  client: nova
  name: Server
  icon: fa:server
  model:
    stack:
      type: relationship_to
      model: in_os_stack
      target: os_stack
    project:
      type: relationship_to
      model: in_os_project
      target: os_project
    network:
      type: relationship_to
      model: use_os_port
      target: os_port
    hypervisor:
      type: relationship_to
      model: on_os_hypervisor
      target: os_hypervisor
```

(continues on next page)

(continued from previous page)

```
    key_pair:
      type: relationship_to
      model: use_os_key_pair
      target: os_key_pair
    image:
      type: relationship_to
      model: use_os_image
      target: os_image
    flavor:
      type: relationship_to
      model: use_os_flavor
      target: os_flavor
os_stack:
  resource: OS::Heat::Stack
  client: heat
  name: Stack
  icon: fa:cubes
  model:
    project:
      type: relationship_to
      model: in_os_project
      target: os_project
os_subnet:
  resource: OS::Neutron::Subnet
  client: neutron
  name: Subnet
  icon: fa:share-alt
  model:
    stack:
      type: relationship_to
      model: in_os_stack
      target: os_stack
    project:
      type: relationship_to
      model: in_os_project
      target: os_project
    network:
      type: relationship_to
      model: in_os_network
      target: os_network
os_user:
  resource: OS::Keystone::User
  client: keystone
  name: User
  icon: fa:cube
os_volume:
  resource: OS::Cinder::Volume
  client: cinder
  name: Volume
  icon: fa:cube
  model:
    stack:
      type: relationship_to
      model: in_os_stack
      target: os_stack
    project:
      type: relationship_to
```

(continues on next page)

(continued from previous page)

```
        model: in_os_project
        target: os_project
```

## 3.3 Container Resource Management

### 3.3.1 Kubernetes Clusters

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

```
scope: global
cluster:
  certificate-authority-data: |
    {{ ca-for-server-and-clients }}
  server: https://{{ kubernetes-api }}:443
user:
  client-certificate-data: |
    {{ client-cert-public }}
  client-key-data: |
    {{ client-cert-private }}
```

**Note:** Options `config.cluster` and `config.user` can be found in your `kubeconfig` file. Just copy the config fragment with cluster parameters and fragment with user parameter.

The metadata schema for Kubernetes manager:

```
query:
  k8s_complete_graph:
    name: Complete Cluster
    layout: graph
  k8s_pods_in_namespaces_tree:
    name: Pods by Namespace
    layout: hierarchy
    hierarchy_layers:
      0:
        name: Kubernetes
        kind:
      1:
        kind: k8s_namespace
      2:
        kind: k8s_pod
        target: in_k8s_namespace
      3:
        kind: k8s_container
        source: in_k8s_pod
  k8s_pods_on_nodes_tree:
    name: Pods by Node
    layout: hierarchy
    hierarchy_layers:
      0:
        name: Kubernetes
```

(continues on next page)

(continued from previous page)

```
        kind:
      1:
        kind: k8s_node
      2:
        kind: k8s_pod
        target: on_k8s_node
relation:
  in_k8s_namespace:
    relation:
      default: namespace
  in_k8s_deployment:
    relation:
      default: deployment
  expose_k8s_pod:
    relation:
      default: pod

  in_k8s_pod:
    relation:
      default: pod
  use_k8s_replication:
    relation:
      default: replication_control
      k8s_replication_controller: pod
      k8s_replica_set: pod
  use_k8s_secret:
    relation:
      default: secret
  on_k8s_node:
    relation:
      default: node

  contains_k8s_node:
    relation:
      default: node
  uses_k8s_config_map:
    relation:
      default: config_map
  contains_k8s_namespace:
    relation:
      default: namespace
default_resource: k8s_namespace
resource:
  k8s_config_map:
    client: ''
    icon: fa:file-text-o
    name: Config Map
    resource: ConfigMap
  k8s_container:
    client: ''
    icon: fa:cube
    name: Container
    resource: Container
    model:
      pod:
        type: relationship_to
        model: in_k8s_pod
```

(continues on next page)

(continued from previous page)

```
      target: k8s_pod
k8s_cron_job:
  client: ''
  icon: fa:cube
  name: Cron Job
  resource: CronJob
k8s_deployment:
  client: ''
  icon: fa:cubes
  name: Deployment
  resource: Deployment
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
      target: k8s_namespace
k8s_endpoint:
  client: ''
  icon: fa:cube
  name: Endpoint
  resource: Endpoint
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
      target: k8s_namespace
k8s_event:
  client: ''
  icon: fa:cube
  name: Event
  resource: Event
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
      target: k8s_namespace
k8s_job:
  client: ''
  icon: fa:cube
  name: Job
  resource: Job
k8s_namespace:
  client: ''
  icon: fa:cube
  name: Namespace
  resource: Namespace
k8s_node:
  client: ''
  icon: fa:server
  name: Node
  resource: Node
k8s_persistent_volume:
  client: ''
  icon: fa:hdd-o
  name: Persistent Volume
  resource: PersistentVolume
k8s_persistent_volume_claim:
```

(continues on next page)

(continued from previous page)

```
    client: ''
    icon: fa:hdd-o
    name: Persistent Volume Claim
    resource: PersistentVolumeClaim
    model:
      namespace:
        type: relationship_to
        model: in_k8s_namespace
        target: k8s_namespace
  k8s_pod:
    client: ''
    icon: fa:cubes
    name: Pod
    resource: Pod
    model:
      namespace:
        type: relationship_to
        model: in_k8s_namespace
        target: k8s_namespace
      node:
        type: relationship_to
        model: on_k8s_node
        target: k8s_node
  k8s_replica_set:
    client: ''
    icon: fa:cubes
    name: Replica Set
    resource: ReplicaSet
    model:
      pod:
        type: relationship_from
        model: use_k8s_replication
        target: k8s_pod
      namespace:
        type: relationship_to
        model: in_k8s_namespace
        target: k8s_namespace
      deployment:
        type: relationship_to
        model: in_k8s_deployment
        target: k8s_deployment
  k8s_replication_controller:
    client: ''
    icon: fa:cubes
    name: Replication Controller
    resource: ReplicationController
    model:
      pod:
        type: relationship_from
        model: use_k8s_replication
        target: k8s_pod
      namespace:
        type: relationship_to
        model: in_k8s_namespace
        target: k8s_namespace
      deployment:
        type: relationship_to
```

(continues on next page)

(continued from previous page)

```
      model: in_k8s_deployment
      target: k8s_deployment
k8s_role:
  client: ''
  icon: fa:cube
  name: Role
  resource: Role
k8s_secret:
  client: ''
  icon: fa:lock
  name: Secret
  resource: Secret
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
      target: k8s_namespace
k8s_service:
  client: ''
  icon: fa:podcast
  name: Service
  resource: Service
  workflow:
    discover_service:
      name: Discover service
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
      target: k8s_namespace
    pod:
      type: relationship_to
      model: expose_k8s_pod
      target: k8s_pod
k8s_service_account:
  client: ''
  icon: fa:user
  name: Service Account
  resource: ServiceAccount
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
      target: k8s_namespace
    secret:
      type: relationship_to
      model: use_k8s_secret
      target: k8s_secret
k8s_ingress:
  client: ''
  icon: fa:user
  name: Ingress
  resource: Ingress
  model:
    namespace:
      type: relationship_to
      model: in_k8s_namespace
```

(continues on next page)

(continued from previous page)

```
      target: k8s_namespace
  k8s_daemon_set:
    client: ''
    icon: fa:user
    name: Daemon Set
    resource: DaemonSet
    model:
      namespace:
        type: relationship_to
        model: in_k8s_namespace
        target: k8s_namespace
  k8s_cluster:
    client: ''
    icon: fa:user
    name: Cluster
    resource: Cluster
    workflow:
      download_config:
        name: Download Config
    model:
      namespaces:
        type: relationship_to
        model: contains_k8s_namespace
        target: k8s_namespace
      nodes:
        type: relationship_to
        model: contains_k8s_node
        target: k8s_node
```

## 3.4 Template Based Orchestration

### 3.4.1 Heat Templates

Heat templates with local context (env) definitions.

```
engine: heat
cloud_endpoint: {{ openstack-manager-name }}
root_path: /path/to/heat-templates
template_path: /path/to/heat-templates/template
context_source: local
context_path: /path/to/heat-templates/env
```

Heat templates with remote context (env) definitions coming from Inventory service.

```
engine: heat
cloud_endpoint: {{ openstack-manager-name }}
root_path: /path/to/heat-templates
template_path: /path/to/heat-templates/template
context_source: remote
context_inventory: {{ inventory-name }}
```

The metadata schema for Heat manager:

```
query:
  heat_complete_graph:
    name: Complete Stack
    layout: graph
  heat_template_stacks_tree:
    name: Template Stacks
    layout: hierarchy
    hierarchy_layers:
      0:
        name: Heat heat_template_stacks_tree
        kind:
      1:
        kind: heat_template
      2:
        kind: heat_stack
        target: defined_by
relation:
  defined_by:
    relation:
      default: template
default_resource: heat_template
resource:
  heat_template:
    client: heat
    icon: fa:cube
    name: Template
    resource: OS::Heat::Stack
    workflow:
      create:
        name: Launch stack
  heat_stack:
    client: heat
    icon: fa:cube
    name: Stack
    resource: OS::Heat::Stack
    model:
      template:
        type: relationship_to
        model: defined_by
        target: heat_template
```

#### References

- https://docs.openstack.org/heat/pike/template_guide/openstack.html
- https://github.com/openstack/heat-templates

### 3.4.2 TerraForm Templates

Configuration for parsing Hashicorp TerraForm templates.

```
monitor:
  terraform-local-manager:
    engine: terraform
    endpoints:
```

(continues on next page)

(continued from previous page)

```
    - {{ kubernetes-manager-name }}
    - {{ openstack-manager-name }}
    - {{ amazon-manager-name }}
    terraform_bin: /path/to/terraform-bin
    template_path: /path/to/terraform-templates
```

The metadata schema for Terraform manager:

```
query:
  tf_complete_graph:
    name: Complete Template
    layout: graph
relation:
  uses_tf_template:
    relation:
      default: template
  in_tf_state:
    relation:
      default: state
  in_tf_module:
    relation:
      default: module
  in_tf_net:
    relation:
      default: network
  in_tf_subnet:
    relation:
      default: subnet
  has_tf_security_group:
    relation:
      default: security_group
  using_tf_key_pair:
    relation:
      default: key_pair
  links_tf_router:
    relation:
      default: router
  links_tf_floating_instance:
    relation:
      default: instance
  links_tf_floating_ip:
    relation:
      tf_openstack_compute_floatingip_associate_v2: floating_ip
      tf_openstack_networking_floatingip_v2: router
default_resource: tf_template
resource:
  tf_template:
    client: ''
    icon: fa:map-signs
    name: Template
    resource: ''
    workflow:
      create:
        name: Deploy template
  tf_state:
    client: ''
    icon: fa:map-signs
```

(continues on next page)

(continued from previous page)

```
    name: State
    resource: ''
    workflow:
      destroy:
        name: Destroy
    model:
      template:
        type: relationship_to
        model: uses_tf_template
        target: tf_template
  tf_module:
    client: ''
    icon: fa:map-signs
    name: Module
    resource: ''
    model:
      state:
        type: relationship_to
        model: in_tf_state
        target: tf_state
  tf_openstack_compute_floatingip_associate_v2:
    client: ''
    icon: fa:map-signs
    name: Floating IP Association
    resource: openstack_compute_floatingip_associate_v2
    model:
      module:
        type: relationship_to
        model: in_tf_module
        target: tf_module
      floating_ip:
        type: relationship_to
        model: links_tf_floating_ip
        target: tf_openstack_networking_floatingip_v2
      instance:
        type: relationship_to
        model: links_tf_floating_instance
        target: tf_openstack_compute_instance_v2
  tf_openstack_compute_instance_v2:
    client: ''
    icon: fa:server
    name: Instance
    resource: openstack_compute_instance_v2
    model:
      module:
        type: relationship_to
        model: in_tf_module
        target: tf_module
      key_pair:
        type: relationship_to
        model: using_tf_key_pair
        target: tf_openstack_compute_keypair_v2
      security_group:
        type: relationship_to
        model: has_tf_security_group
        target: tf_openstack_compute_secgroup_v2
      network:
```

(continues on next page)

(continued from previous page)

```
      type: relationship_to
      model: in_tf_net
      target: tf_openstack_networking_network_v2
tf_openstack_compute_keypair_v2:
  client: ''
  icon: fa:key
  name: Key Pair
  resource: openstack_compute_keypair_v2
tf_openstack_compute_secgroup_v2:
  client: ''
  icon: fa:cube
  name: Security Group
  resource: openstack_compute_secgroup_v2
  model:
    module:
      type: relationship_to
      model: in_tf_module
      target: tf_module
tf_openstack_networking_floatingip_v2:
  client: ''
  icon: fa:map-signs
  name: Floating IP
  resource: openstack_networking_floatingip_v2
  model:
    module:
      type: relationship_to
      model: in_tf_module
      target: tf_module
    router:
      type: relationship_to
      model: links_tf_floating_ip
      target: tf_openstack_networking_router_interface_v2
tf_openstack_networking_network_v2:
  client: ''
  icon: fa:share-alt
  name: Net
  resource: openstack_networking_network_v2
  model:
    module:
      type: relationship_to
      model: in_tf_module
      target: tf_module
    key_pair:
      type: relationship_to
      model: in_tf_net
      target: tf_openstack_networking_network_v2
tf_openstack_networking_router_interface_v2:
  client: ''
  icon: fa:arrows-alt
  name: Router Interface
  resource: openstack_networking_router_interface_v2
  model:
    module:
      type: relationship_to
      model: in_tf_module
      target: tf_module
    subnet:
```

(continues on next page)

(continued from previous page)

```
      type: relationship_to
      model: in_tf_subnet
      target: tf_openstack_networking_subnet_v2
    router:
      type: relationship_to
      model: links_tf_router
      target: tf_openstack_networking_router_v2
tf_openstack_networking_router_v2:
  client: ''
  icon: fa:arrows-alt
  name: Router
  resource: openstack_networking_router_v2
  model:
    module:
      type: relationship_to
      model: in_tf_module
      target: tf_module
tf_openstack_networking_subnet_v2:
  client: ''
  icon: fa:share-alt
  name: Subnet
  resource: openstack_networking_subnet_v2
  model:
    module:
      type: relationship_to
      model: in_tf_module
      target: tf_module
    network:
      type: relationship_to
      model: in_tf_net
      target: tf_openstack_networking_network_v2
```

### References

- https://www.terraform.io/docs/index.html
- https://github.com/terraform-providers

CHAPTER 4

# Architect Monitor

## 4.1 Time-series Monitoring

Monitor components can query metrics from several time-series databases into uniform Pandas DataFrames.

It support two types of metric queries, the first is `instant` metric, returning the value in precise moment in time. The second is the `range` metric, giving you the series of values for given time range and step.

The Monitor supports several major time-series databases to get the results in normalised way. The endpoints are queried thru HTTP API calls.

### 4.1.1 Graphite Time-series Database

Example configuration for the Graphite server.

```
monitor:
  graphite-inventory:
    auth_url: http://{{ graphite-api }}:8000
```

Example query to the Graphite server.

```
averageSeries(server.web*.load)
```

The metadata schema for Graphite monitor:

```
query:
  prometheus_test_range_query_long:
    metric: hdd_errors
    step: 30m
    start: "2018-01-25T12:00:00Z"
    end: "2018-01-30T12:00:00Z"
  prometheus_test_range_query:
    metric: hdd_errors
```

(continues on next page)

(continued from previous page)

```
    step: 4h
    start: "2018-01-25T12:00:00Z"
    end: "2018-01-30T12:00:00Z"
  prometheus_test_instant_query:
    metric: hdd_errors
    moment: "2018-01-28T12:00:00Z"
metric:
  hdd_errors:
    job: carbon
    name: HDD errors
```

#### References

- http://graphite.readthedocs.io/en/latest/render_api.html

### 4.1.2 InfluxDB Time-series Database

Example configuration for the InfluxDB server.

```
monitor:
  influxdb-inventory:
    auth_url: http://{{ influxdb-api }}:8086
    username: {{ influxdb-username }}
    password: {{ influxdb-password }}
    database: {{ influxdb-database }}
```

Example query to the InfluxDb server.

```
SELECT mean("value") FROM "alertmanager_notifications_total"
```

#### References

- https://docs.influxdata.com/influxdb/v1.3/guides/querying_data/

### 4.1.3 Prometheus Server

Example configuration for the Prometheus server.

```
monitor:
  prometheus-inventory:

    auth_url: http://{{ prometheus-api }}:8000
```

Example query to the Prometheus server.

```
alertmanager_notifications_total
```

The metadata schema for Prometheus monitor:

```
query:
  prometheus_test_range_query_long:
    query: hdd_errors
    step: 30m
    start: "2018-01-25T12:00:00Z"
    end: "2018-01-30T12:00:00Z"
  prometheus_test_range_query:
    query: hdd_errors
    step: 4h
    start: "2018-01-25T12:00:00Z"
    end: "2018-01-30T12:00:00Z"
  prometheus_test_instant_query:
    query: hdd_errors
    moment: "2018-01-28T12:00:00Z"
default_resource: prom_target
relation:
  by_job:
    relation:
      default: prom_job
  metric_value:
    relation:
      default: prom_target
resource:
  prom_metric:
    client: prometheus/series
    icon: fa:cube
    name: Metric
    resource: Prometheus::Series
    workflow:
      display_metric:
        name: Display chart
    model:
      target:
        type: relationship_to
        model: metric_value
        target: prom_target
  prom_target:
    client: prometheus/targets
    icon: fa:cube
    name: Target
    resource: Prometheus::Target
    model:
      job:
        type: relationship_to
        model: by_job
        target: prom_job
  prom_job:
    client: prometheus/job
    icon: fa:cube
    name: Job
    resource: Prometheus::Job
```

### References

- https://prometheus.io/docs/prometheus/latest/querying/api/
- https://github.com/infinityworks/prometheus-example-queries

CHAPTER 5

# Architect Repository

## 5.1 Image Building Overview

The architect is capable of pre-building images with specific content by SaltStack. For this you need to setup the metadata model of the node and select the Salt Master server. The procedure of building is:

1. Create the new inventory object (node definition), the `reclass` and `cluster-deploy` inventory types are supported.
2. Build the new image for the selected platform and node definition. The node does not need to be registered at Salt master, it can be added in process.

## 5.2 BeagleBone/BeagleBoard Images

The BeagleBoard image builder used is fork of official Beagle image builder. To install this image builder, clone repo https://github.com/salt-formulas/beagleboard-image-builder to the location specified in repository configuration.

```
repository:
  bbb-repo:
    engine: bbb
    builder_dir: /path/to/bbb-image-build
    image_dir: /storage/dir
    manager: salt-manager
    inventory: reclass-inventory
```

### 5.2.1 Build Script

```
#!/bin/bash -e

# Usage:
```

(continues on next page)

(continued from previous page)

```
#
# ./gen-image.sh node.domain.com-20180405 node.domain.com bbb
#
# bbb  - BeagleBone Black Rev B/C, BeagleBone Blue
# bbx15 - BeagleBoard-X15

IMAGENAME="${1:-rpi.domain-config-datetime}"
HOSTNAME="${2:-rpi.domain-config}"
PLATFORM="${3:-bbb}"
OUTPUT="${4:-noop}"

base_rootfs="$IMAGENAME"
wfile="$IMAGENAME"

DIR="$PWD"

archive_base_rootfs () {
        if [ -d ./${base_rootfs} ] ; then
                rm -rf ${base_rootfs} || true
        fi
        if [ -f ${base_rootfs}.tar ] ; then
                xz -z -8 -v ${base_rootfs}.tar && sha256sum ${base_rootfs}.tar.xz > ${base_rootfs}.tar.xz.sha256sum &
        fi
}

extract_base_rootfs () {
        if [ -d ./${base_rootfs} ] ; then
                rm -rf ${base_rootfs} || true
        fi

        if [ -f ${base_rootfs}.tar.xz ] ; then
                tar xf ${base_rootfs}.tar.xz
        else
                tar xf ${base_rootfs}.tar
        fi
}

archive_img () {
        #prevent xz warning for 'Cannot set the file group: Operation not permitted'
        sudo chown ${UID}:${GROUPS} ${wfile}.img
        if [ -f ${wfile}.img ] ; then
                if [ ! -f ${wfile}.bmap ] ; then
                        if [ -f /usr/bin/bmaptool ] ; then
                                bmaptool create -o ${wfile}.bmap ${wfile}.img
                        fi
                fi
                xz -z -8 -v ${wfile}.img && sha256sum ${wfile}.img.xz > ${wfile}.img.xz.sha256sum &
        fi
}

generate_img () {
        cd ${base_rootfs}/
        sudo ./setup_sdcard.sh ${options}
        mv *.img ../
        cd ..
```

(continues on next page)

(continued from previous page)

```
}

./RootStock-NG.sh -c $IMAGENAME

if [ "$PLATFORM" = "bbb" ] ; then
        platform_options="--dtb beaglebone --bbb-old-bootloader-in-emmc --emmc-flasher
↪"
fi

if [ "$PLATFORM" = "bbx15" ] ; then
        platform_options="--dtb am57xx-beagle-x15"
fi

options="--img ${IMAGENAME} ${platform_options} --hostname ${HOSTNAME}"

cd ./deploy

extract_base_rootfs
generate_img
archive_base_rootfs
archive_img

if [ -f ${wfile}.img ] ; then
        mv ./${wfile}.img ${OUTPUT}
fi

if [ -f ${wfile}.bmap ] ; then
        mv ./${wfile}.bmap ${OUTPUT}
fi

cd ..
```

## 5.3 RaspberryPi Images

The Raspberry Pi image builder used is fork of official Beagle image builder. To install this image builder, clone repo https://github.com/salt-formulas/rpi23-gen-image to the location specified in repository configuration.

```
repository:
  bbb-repo:
    engine: rpi23
    builder_dir: /path/to/rpi23-gen-image
    image_dir: /storage/dir
    manager: salt-manager
    inventory: reclass-inventory
```

### 5.3.1 Build Script

The `rpi23-gen-image` uses modified generator script.

```
#!/bin/sh

# Usage:
```

(continues on next page)

(continued from previous page)

```
#
# ./gen-image.sh node.domain.com-20180405

set -e
set -x

IMAGENAME="${1:-rpi.domain.com-timestamp}"

rm -rf "./images/jessie/"
rm -rf "./images/stretch/"
rm -rf "./images/buster/"

CONFIG_TEMPLATE="${IMAGENAME}" ./rpi23-gen-image.sh

rm -rf "./images/jessie/"
rm -rf "./images/stretch/"
rm -rf "./images/buster/"
```

CHAPTER 6

# Architect Document

## 6.1 Visual Infographics

It is useful to identify three main categories of data visualizations in terms of what their main (intended or unintended) purpose is.

**Inspirational**

The main goal here is to inspire people. To *wow* them! But not just on a superficial level, but to really engage people into deeper thinking, sense of beauty and awe. Visualization has an incredible power to attract people's attention but also to draw them into fantastic artificial worlds that turn abstract concept into more tangible ones.

**Explanatory**

The main goal here is to use graphics as a way to explain some complex idea, phenomenon or process. This is an area where graphical representation shines: we are visual creatures and a picture is sometime really worth a thousand words.

**Analytical**

The main goal here is to extract information out of data with the purpose of answering questions and advancing understanding of some phenomenon of interest. Sure, explanatory visualization is also about helping people understand something.

But the main difference here is that in *explanatory visualization* the author knows already what to visualize (after having performed some analysis), whereas in analysis the main use of visualization is to understand the data in the first place.

Data analysis is important because it can help people improve their understanding of complex phenomena in our case service models and can help solve important problems around it. It's an indirect link, but an important one: *If I understand a problem better, there are higher chances I can find a better solution for it*.

### 6.1.1 Core Presentation Libraries

#### d3.js

D3 (or D3.js) is a JavaScript library for visualizing data using web standards. D3 helps you bring data to life using SVG, Canvas and HTML. D3 combines powerful visualization and interaction techniques with a data-driven approach to DOM manipulation, giving you the full capabilities of modern browsers and the freedom to design the right visual interface for your data.

- https://github.com/d3/d3
- http://bost.ocks.org/mike/chart/ Towards Reusable Charts
- https://bocoup.com/blog/reusability-with-d3 Exploring Reusability with D3.js
- https://github.com/springload/react-d3-integration An example on how to integrate D3 into React
- http://bl.ocks.org/biovisualize/8187844 direct svg to canvas to png conversion
- http://bl.ocks.org/vicapow/758fce6aa4c5195d24be An example of creating a PNG from an SVG in D3.

### 6.1.2 Presentation Helper Libraries

#### d3 layouts

- http://blog.visual.ly/cartesian-vs-radial-charts/ Battle of the Charts: Why Cartesian Wins Against Radial
- http://bl.ocks.org/giuliano108/7482331 d3.layout.colgrid
- https://bl.ocks.org/feyderm/ba5a80beec95ff39b5267554b590993f A change of perspective...

#### d3-legend

A library to make graph legends.

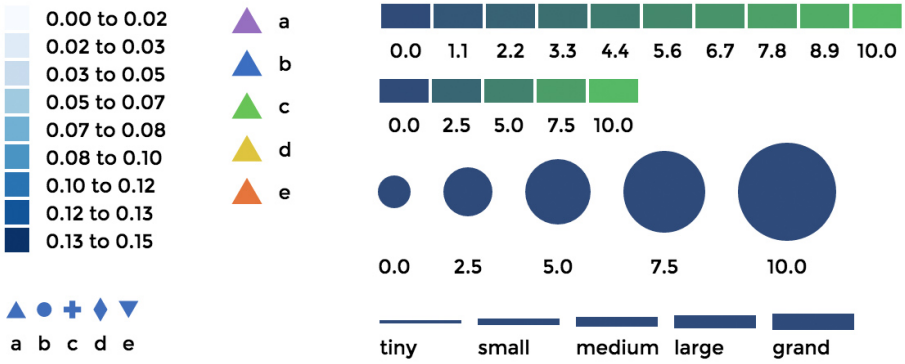



Fig. 1: d3-legend diagram

- https://github.com/susielu/d3-legend
- http://d3-legend.susielu.com/

#### d3-annotation

Annotations establish context, and direct our users to insights and anomalies.

- https://github.com/susielu/d3-annotation
- http://d3-annotation.susielu.com/
- https://bl.ocks.org/alansmithy/85e2d6e05f1de59167751249fbd1edec

## 6.2 Relational Data Analysis

You can analyse the resource models in several ways. Either you want to get the subsets of the resources (vertices and edges) or you want to combine multiple graphs and link the same nodes in each.

### 6.2.1 Relational Schema

All resources are covered by schema that define basic properties of the nodes and relationships.

#### Resource Nodes

Sample node definition of the OpenStack hypervisor resource. We declare directional typed relationships at either side by `relationship_to` and `relationship_from` parameters.

```
os_hypervisor:
  resource: OS::Nova::Hypervisor
  client: nova
  name: Hypervisor
  icon: fa:server
  model:
    aggregate:
      type: relationship_to
      model: in_os_aggregate
      target: os_aggregate
```

#### Resource Relations

Along the node definition we define the relations.

```
on_os_hypervisor:
  relation:
    os_server: hypervisor
    os_port: hypervisor
in_os_aggregate:
  relation:
    os_hypervisor: aggregate
```

### 6.2.2 Relational Operations

We can either break a body of information down into smaller parts or to examine it from different viewpoints that we can understand it better and we can also combine multiple bodies in one get further insight.

### Subgraphs - Slicing and Dicing

To slice and dice is to break a body of information down into smaller parts or to examine it from different viewpoints that we can understand it better.

In cooking, you can slice a vegetable or other food or you can dice it (which means to break it down into small cubes). One approach to dicing is to first slice and then cut the slices up into dices.

```
name: Hive-plot
data_source:
  default:
    manager: openstack-project
    layout: graph
    filter_node_types:
    - os_server
    - os_key_pair
    - os_flavor
    - os_network
    - os_subnet
    - os_floating_ip
    - os_router
    filter_lone_nodes:
    - os_key_pair
    - os_flavor
```

In data analysis, the term generally implies a systematic reduction of a body of data into smaller parts or views that will yield more information. The term is also used to mean the presentation of information in a variety of different and useful ways. In our case we find useful subgraphs of the infrastructures.

### Hiearchical Structures

In some cases it is useful to crate hierarchical structures from graph data. For example in OpenStack infrastructure we can show the `aggregate zone` - `hypervisor` - `instance` relations and show the quantitative properties of hypervisors and instances. The properties can be used RAM or CPU, runtime - the age of resources or any other property of value.

```
name: Tree Structure (aggregate zone > hypervisor > instance)
height: 1
chart: tree
data_source:
  default:
    manager: openstack-region
    layout: hierarchy
    hierarchy_layers:
      0:
        name: Region1
        kind:
      1:
        kind: os_aggregate_zone
      2:
        kind: os_hypervisor
        target: in_os_aggregate_zone
      3:
        kind: os_server
        target: on_os_hypervisor
```

Another example would be filtering of resources by tenant or stack attributions. This reduces the number of nodes to the reasonable amount.

#### Inter-graphs

On other hand you want to combine several graphs to create one overlaying graph. This is very useful to combine in other ways undelated resources. For example we can say that `OpenStack Server` or `AWS Instance` and `Salt Minion` are really the same resources.

```
name: Hive-plot
data_source:
  default:
    manager: openstack-project
    layout: graph
    filter_node_types:
    - os_server
```

## 6.3 Quantitative Data Analysis

With the relational information we are now able to corellate resources and joined topologies from varius information sources. This gives you the real power, while having the underlying relational structure, you can gather unstructured metrics, events, alarms and put them into proper context in you managed resources.

The metrics collected from you infrastrucute by means of local monitorin system can be assigned to various vertices and edges in your network. This can give you more insight to the utilisation of depicted infrastructures.

### 6.3.1 Query Options

#### Time-series Metrics

Parameters that apply only for the `range` metrics.

**start** Time range start.

**end** Time range end.

**step** Query resolution step width.

#### Instant Metric

Parameters that apply only for the `intant` meters.

**moment** Single moment in time.

### 6.3.2 Alarm Options

Following lists show allowed values for alarm functions, the alarm arithmetic operators and aggregation function for `range` meters.

#### Supported Time-series Aggregations

**avg** Arithmetic average of the series values.

**min** Use the minimal value from series.

**max** Use the maximal value from series.

**sum** Sum the values together.

### 6.3.3 Advanced Usage

You can have the following query to the prometheus server that gives you the rate of error response codes goint through a HAproxy for example.

```
sum(irate(haproxy_http_response_5xx{
    proxy=~"glance.*",
    sv="FRONTEND"
}[5m]))
```

Or you can have the query with the same result to the InfluxDB server:

```
SELECT sum("count")
    FROM "openstack_glance_http_response_times"
    WHERE "hostname" =~ /$server/
        AND "http_status" = '5xx'
        AND $timeFilter
    GROUP BY time($interval)
fill(0)
```

Having these metrics you can assign numerical properties of your relational nodes with these values and use them in correct context.

## 6.4 Network Graph Visualizations

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.

### 6.4.1 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),

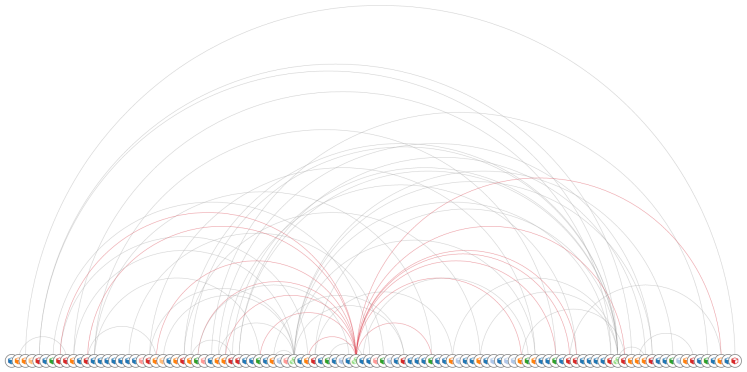


Fig. 2: OpenStack project resources in Arc diagram (cca 100 resources)

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.

#### References

- https://en.wikipedia.org/wiki/Arc_diagram
- https://bl.ocks.org/rpgove/53bb49d6ed762139f33bdaea1f3a9e1c Arc diagram
- http://bl.ocks.org/sjengle/5431779 D3 Arc Diagram

### 6.4.2 Force-Directed Graph

A *Force-directed graph* drawing algorithms are used for drawing graphs in an aesthetically pleasing way. Their purpose is to position the nodes of a graph in two-dimensional or three-dimensional space so that all the edges are of more or less equal length and there are as few crossing edges as possible, by assigning forces among the set of edges and the set of nodes, based on their relative positions, and then using these forces either to simulate the motion of the edges and nodes or to minimize their energy.

While graph drawing can be a difficult problem, force-directed algorithms, being physical simulations, usually require no special knowledge about graph theory such as planarity.

Good-quality results can be achieved for graphs of medium size (up to 50–500 vertices), the results obtained have usually very good results based on the following criteria: uniform edge length, uniform vertex distribution and showing symmetry. This last criterion is among the most important ones and is hard to achieve with any other type of algorithm.

#### References

- https://en.wikipedia.org/wiki/Force-directed_graph_drawing
- https://bl.ocks.org/shimizu/e6209de87cdddde38dadbb746feaf3a3 shimizu's D3 v4 - force layout
- https://bl.ocks.org/mbostock/3750558 Mike Bostock's Sticky Force Layout
- https://bl.ocks.org/emeeks/302096884d5fbc1817062492605b50dd D3v4 Constraint-Based Layout

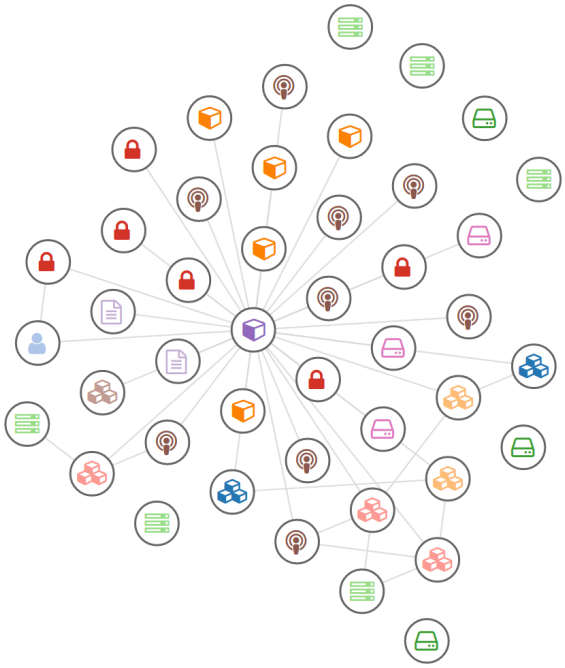


Fig. 3: Kubernetes cluster in Force-directed graph

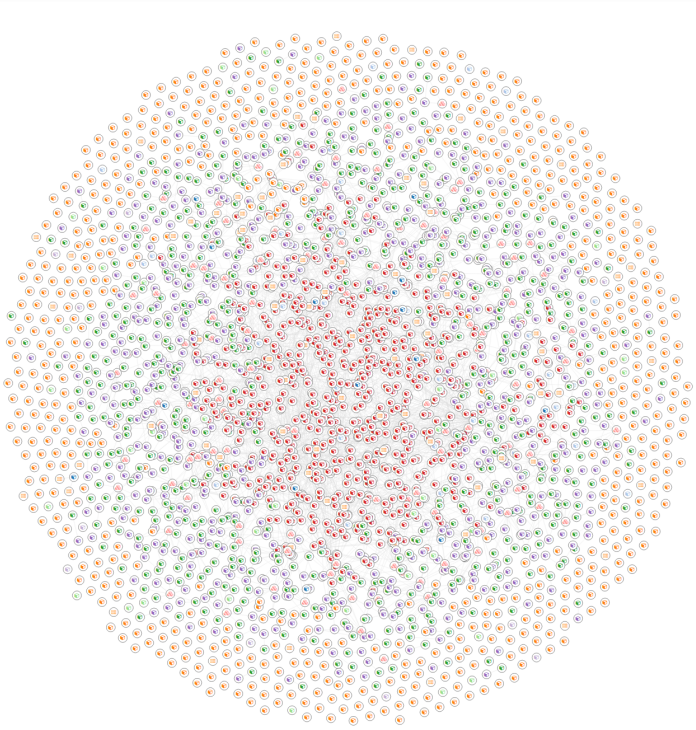


Fig. 4: Whole OpenStack cloud in Force-directed graph (cca 3000 resources)

- http://bl.ocks.org/biovisualize/5801758 dag layout
- http://bl.ocks.org/bobbydavid/5841683 DAG visualization
- https://bl.ocks.org/emeeks/302096884d5fbc1817062492605b50dd D3v4 Constraint-Based Layout
- https://bl.ocks.org/denisemauldin/cdd667cbaf7b45d600a634c8ae32fae5 Filtering Nodes on Force-Directed Graphs (D3 V4)

### 6.4.3 Hierarchical Edge Bundling

A *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.

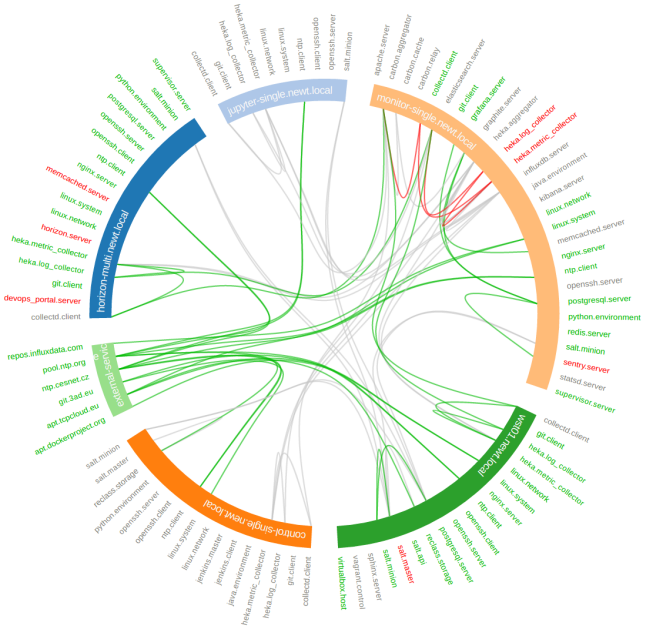



Fig. 5: Hierarchical edge bundling of SaltStack services and their relations (cca 100 nodes)

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.

#### References

- http://www.win.tue.nl/vis1/home/dholten/papers/bundles_infovis.pdf
- https://www.win.tue.nl/vis1/home/dholten/papers/forcebundles_eurovis.pdf
- https://bl.ocks.org/mbostock/7607999 Hierarchical Edge Bundling

### 6.4.4 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.

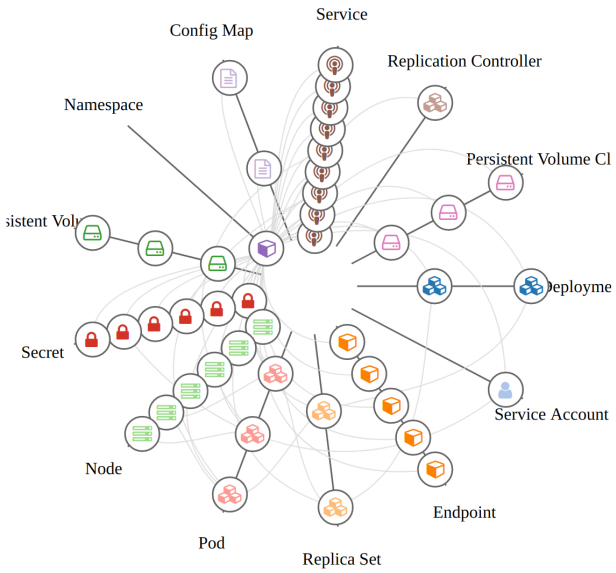



Fig. 6: Kubernetes cluster in Hive plot

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.

### References

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

## 6.4.5 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.

### References

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

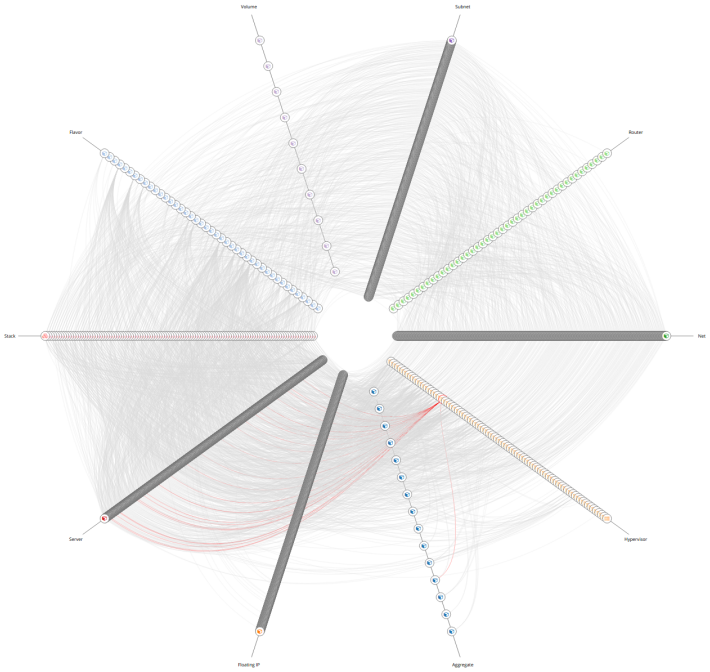



Fig. 7: Whole OpenStack cloud in Hive plot (cca 10 000 resources)

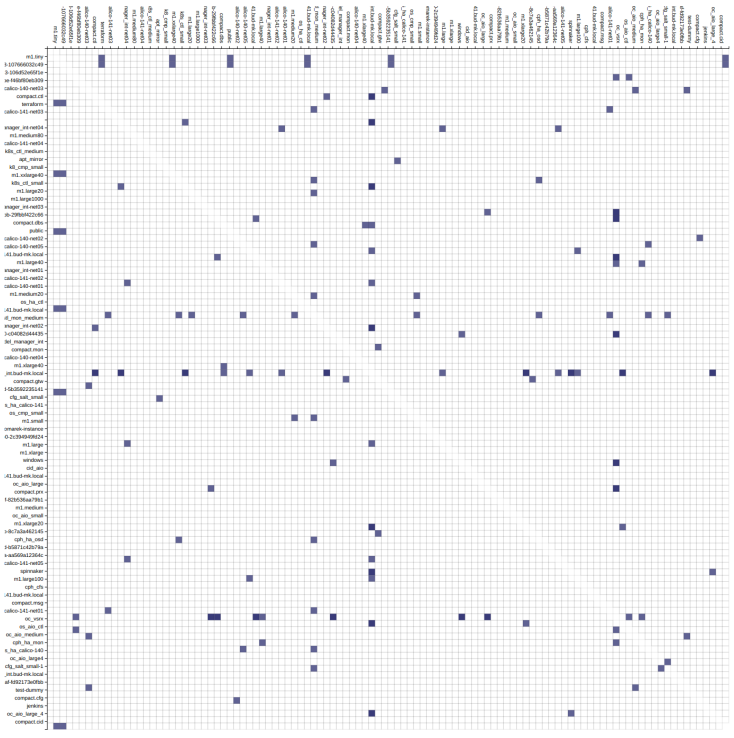


Fig. 8: Adjacency matrix of OpenStack project's resources (cca 100 nodes)

### 6.4.6 Sankey Diagram

*Sankey diagrams* are a specific type of flow diagram, in which the width of the arrows is shown proportionally to the flow quantity. Sankey diagrams put a visual emphasis on the major transfers or flows within a system. They are helpful in locating dominant contributions to an overall flow. Often, Sankey diagrams show conserved quantities within defined system boundaries.

Sankey diagrams are named after Irish Captain Matthew Henry Phineas Riall Sankey, who used this type of diagram in 1898 in a classic figure (see panel on the right) showing the energy efficiency of a steam engine. While the first charts in black and white were merely used to display one type of flow (e.g. steam), using colors for different types of flows has added more degrees of freedom to Sankey diagrams.

One of the most famous Sankey diagrams is Charles Minard's Map of Napoleon's Russian Campaign of 1812. It is a flow map, overlaying a Sankey diagram onto a geographical map. It was created in 1869, so it actually predates Sankey's 'first' Sankey diagram of 1898.

#### References

- https://en.wikipedia.org/wiki/Sankey_diagram
- https://github.com/FabricioRHS/skd3
- https://bl.ocks.org/emeeks/e9d64d27f286e61493c9 Sankey Particles IV

### 6.4.7 Alluvial Diagram

Alluvial diagrams are a type of flow diagram originally developed to represent changes in network structure over time. In allusion to both their visual appearance and their emphasis on flow, alluvial diagrams are named after alluvial fans that are naturally formed by the soil deposited from streaming water.

#### References

- https://en.wikipedia.org/wiki/Alluvial_diagram
- http://bl.ocks.org/igorzilla/3086583 Alluvial Diagram

## 6.5 Hierarchical Visualizations

Tree 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.

### 6.5.1 Dendrogram, Reingold–Tilford Tree

The *dendrograms* are node-link diagrams that place leaf nodes of the tree at the same depth. Dendograms are typically less compact than *tidy trees*, but are useful when all the leaves should be at the same level, such as for hierarchical clustering or phylogenetic tree diagrams.

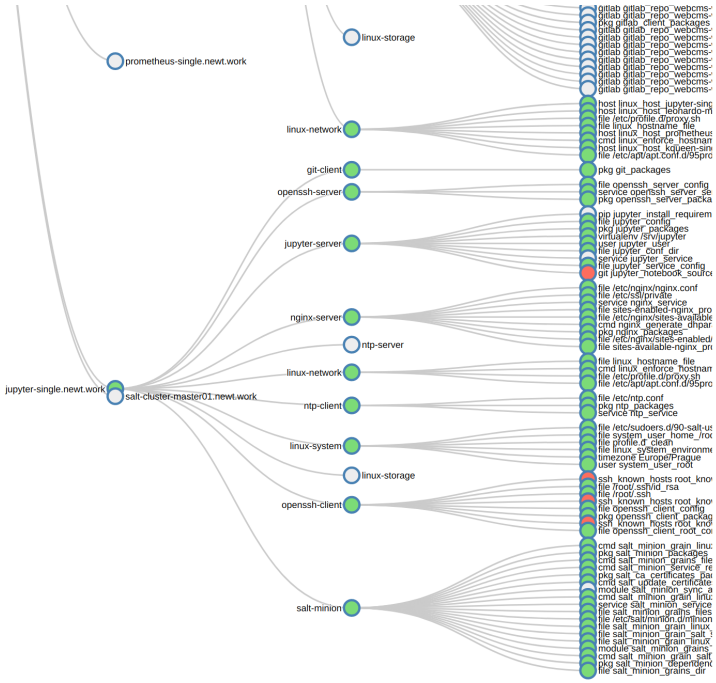



Fig. 9: SaltStack services in Hierarchical edge bundle

### References

- https://en.wikipedia.org/wiki/Dendrogram
- https://en.wikipedia.org/wiki/Radial_tree
- http://ncss.wpengine.netdna-cdn.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Hierarchical_Clustering-Dendrograms.pdf
- http://www.meccanismocomplesso.org/en/dendrogramma-d3-parte1/
- https://bl.ocks.org/mbostock/4063570 Cluster Dendrogram
- http://bl.ocks.org/mbostock/4063550 Radial Reingold–Tilford Tree
- http://bl.ocks.org/mbostock/4339184 Reingold–Tilford Tree

## 6.5.2 Sunburst Chart

A *ring chart*, also known as a *sunburst chart* or a *multilevel pie chart*, is used to visualize hierarchical data, depicted by concentric circles. The circle in the centre represents the root node, with the hierarchy moving outward from the center. A segment of the inner circle bears a hierarchical relationship to those segments of the outer circle which lie within the angular sweep of the parent segment.

The partition layout produces adjacency diagrams: a space-filling variant of a node-link tree diagram. Rather than drawing a link between parent and child in the hierarchy, nodes are drawn as solid areas (either arcs or rectangles), and their placement relative to other nodes reveals their position in the hierarchy. The size of the nodes encodes a quantitative dimension that would be difficult to show in a node-link diagram.

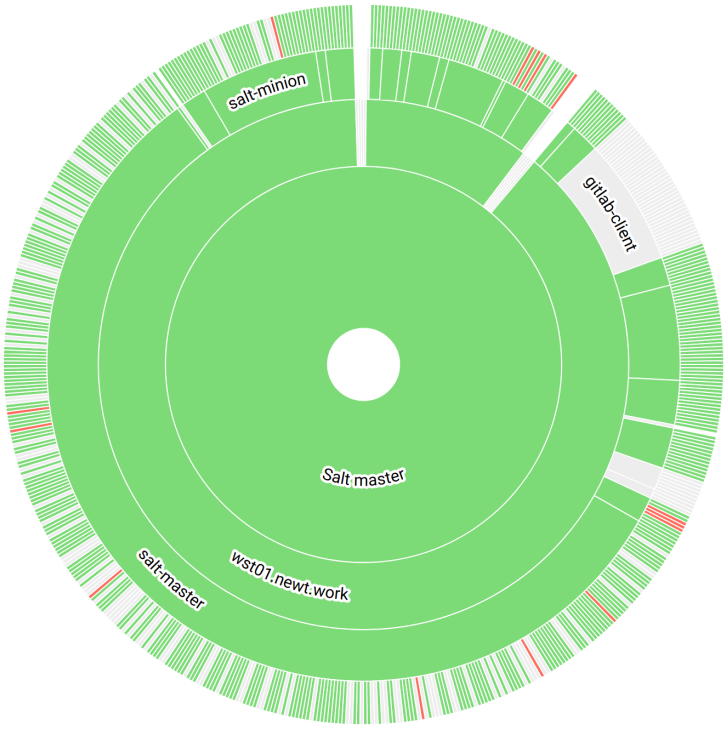



Fig. 10: SaltStack services in Sunburst Diagram

#### References

- https://en.wikipedia.org/wiki/Pie_chart
- https://bl.ocks.org/mbostock/4063423 Sunburst Partition

### 6.5.3 Circle Packing

*Circle packing* in a circle is a two-dimensional packing problem with the objective of packing unit circles into the smallest possible larger circle. Resources lower in hierarchy are diaplayed as circles with lower-level resources as inner circles.

#### References

- https://en.wikipedia.org/wiki/Circle_packing_in_a_circle
- https://bl.ocks.org/mbostock/7607535 Zoomable Circle Packing
- http://bl.ocks.org/vicapow/3d24f96c240eeb8d14e3 circle packing with depth dependent padding

### 6.5.4 Treemap

*Treemap* is a space-constrained visualization of hierarchical structures. It is very effective in showing attributes of leaf nodes using size and color coding. Treemap enables users to compare nodes and sub-trees even at varying depth in the tree, and help them spot patterns and exceptions. Treemap was first designed by Ben Shneiderman during the 1990s.

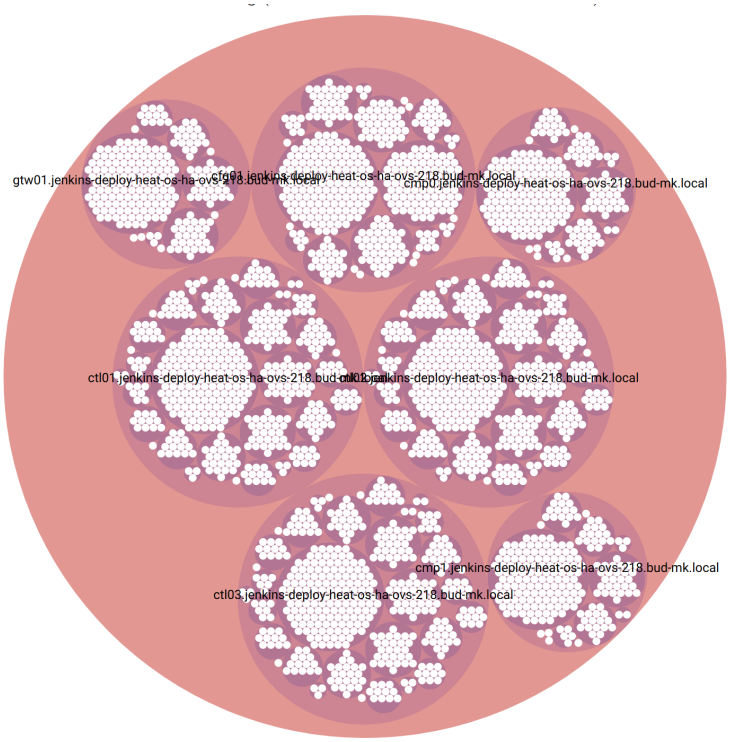



Fig. 11: SaltStack services in Circle Packing

For more information, read the historical summary of treemaps, their growing set of applications, and the many other implementations. Treemaps are a continuing topic of research and application at the HCIL.

When the color and size dimensions are correlated in some way with the tree structure, one can often easily see patterns that would be difficult to spot in other ways, such as if a certain color is particularly relevant. A second advantage of treemaps is that, by construction, they make efficient use of space. As a result, they can legibly display thousands of items on the screen simultaneously.

#### References

- https://en.wikipedia.org/wiki/Treemapping
- https://bl.ocks.org/shimizu/6d60e554dcbba406721e73ed5afdf713 D3 v4 - Treemap
- http://www.cs.umd.edu/hcil/treemap/
- http://www.cs.umd.edu/hcil/treemap-history/
- http://www.billdwhite.com/wordpress/2012/12/16/d3-treemap-with-title-headers/

### 6.5.5 Voronoi Treemap

*Voronoi treemaps* are an alternative to traditional rectangular treemaps for visualizing hierarchical data. Like rectangular treemaps, Voronoi treemaps represent hierarchical data by dividing the canvas region into cells for each node at the top of the hierarchy, and then further dividing each of these cells for the children of those nodes. The organic shapes created by the Voronoi treemap can be easier to distinguish sibling nodes from nodes in other branches of the hierarchy. Voronoi treemaps can also be fit to non-rectangular canvases, and are often more aesthetically pleasing.

#### References

- http://cse512-14w.github.io/fp-plvines-djpeter/
- http://cse512-14w.github.io/fp-plvines-djpeter/demo.html
- http://cse512-14w.github.io/fp-plvines-djpeter/final/paper-plvines-djpeter.pdf

### 6.5.6 Orbital Layout

An animated hierarchical layout that creates orbits from nested data.

- https://github.com/emeeks/d3.layout.orbit
- http://bl.ocks.org/emeeks/068ef3e4106e155467a3 Orbital Layout of D3.js API

## 6.6 DAG Visualizations

A directed acyclic graph (DAG), is a finite directed graph with no directed cycles. That is, it consists of finitely many vertices and edges, with each edge directed from one vertex to another, such that there is no way to start at any vertex v and follow a consistently-directed sequence of edges that eventually loops back to v again. Equivalently, a DAG is a directed graph that has a topological ordering, a sequence of the vertices such that every edge is directed from earlier to later in the sequence.

### 6.6.1 Layered Graph

*Layered graph drawing* or *hierarchical graph drawing* is a type of graph drawing in which the vertices of a directed graph are drawn in horizontal rows or layers with the edges generally directed downwards. It is also known as *Sugiyama-style graph drawing* after Kozo Sugiyama, who first developed this drawing style.

#### References

- https://en.wikipedia.org/wiki/Layered_graph_drawing
- https://github.com/dagrejs/dagre-d3
- https://github.com/jdk137/dag
- https://bl.ocks.org/jebeck/89fd1b6083a19d7f644a A dagre dependency graph
- http://www.samsarin.com/project/dagre-d3/latest/demo/hover.html

## 6.7 Numerical Visualizations

### 6.7.1 Progress Bar

A *progress bar* is a graphical control element used to visualize the progression of an extended computer operation, such as a download, file transfer, or installation. Sometimes, the graphic is accompanied by a textual representation of the progress in a percent format.

The concept of a progress bar was invented before digital computing. In 1896 Karol Adamiecki developed a chart which he called a harmonogram, which is better known today as a Gantt chart. Adamiecki did not publish his chart

until 1931, however, and then only in Polish. The chart thus now bears the name of Henry Gantt (1861–1919), who designed his chart around the years 1910-1915 and popularized it in the west.

#### References

- https://en.wikipedia.org/wiki/Progress_bar
- http://bl.ocks.org/brattonc/d54d1c9d33aa13491279 D3 Bar Stacker Gauge
- http://pablomolnar.github.io/radial-progress-chart/

### 6.7.2 Gauge

A *gauge* or *gage*, in science and engineering, is a device used to make measurements or in order to display certain dimensional information. A wide variety of tools exist which serve such functions, ranging from simple pieces of material against which sizes can be measured to complex pieces of machinery. Depending on usage, a gauge can be described as "a device for measuring and displaying a physical quantity",

#### References

- http://bl.ocks.org/brattonc/5e5ce9beee483220e2f6 D3 Liquid Fill Gauge
- http://bl.ocks.org/bill-kidwell/dc7062e045a11b44fdc80e4c1e47e20f google style gauges using javascript d3.js v4

### 6.7.3 Pie Chart, Doughnut Chart

A *pie chart* (or a *circle chart*) is a circular statistical graphic which is divided into slices to illustrate numerical proportion. In a pie chart, the arc length of each slice (and consequently its central angle and area), is proportional to the quantity it represents. While it is named for its resemblance to a pie which has been sliced, there are variations on the way it can be presented. The earliest known pie chart is generally credited to William Playfair's Statistical Breviary of 1801.

A *3d pie cake*, or *perspective pie cake*, is used to give the chart a 3D look. Often used for aesthetic reasons, the third dimension does not improve the reading of the data; on the contrary, these plots are difficult to interpret because of the distorted effect of perspective associated with the third dimension. The use of superfluous dimensions not used to display the data of interest is discouraged for charts in general, not only for pie charts.

A *doughnut chart* (also spelled *donut*) is a variant of the pie chart, with a blank center allowing for additional information about the data as a whole to be included. Doughnut charts are similar to pie charts in that their aim is to illustrate proportions. This type of circular graph can support multiple statistics at once and it provides a better data intensity ratio to standard pie charts.

The *polar area diagram* is similar to a usual pie chart, except sectors have equal angles and differ rather in how far each sector extends from the center of the circle. The polar area diagram is used to plot cyclic phenomena (e.g., counts of deaths by month). For example, if the counts of deaths in each month for a year are to be plotted then there will be 12 sectors (one per month) all with the same angle of 30 degrees each. The radius of each sector would be proportional to the square root of the death count for the month, so the area of a sector represents the number of deaths in a month. If the death count in each month is subdivided by cause of death, it is possible to make multiple comparisons on one diagram, as is seen in the polar area diagram famously developed by Florence Nightingale.

A *ring chart*, also known as a *sunburst chart* or a *multilevel pie chart*, is used to visualize hierarchical data, depicted by concentric circles. The circle in the centre represents the root node, with the hierarchy moving outward from the

center. A segment of the inner circle bears a hierarchical relationship to those segments of the outer circle which lie within the angular sweep of the parent segment.

A *spie chart* comparing number of students with student costs across four different schools A variant of the polar area chart is the spie chart designed by Dror Feitelson. This superimposes a normal pie chart with a modified polar area chart to permit the comparison of two sets of related data. The base pie chart represents the first data set in the usual way, with different slice sizes. The second set is represented by the superimposed polar area chart, using the same angles as the base, and adjusting the radii to fit the data. For example, the base pie chart could show the distribution of age and gender groups in a population, and the overlay their representation among road casualties. Age and gender groups that are especially susceptible to being involved in accidents then stand out as slices that extend beyond the original pie chart.

The *square charts* are a rare form of pie charts that use squares instead of circles to represent percentages. Similar to basic circular pie charts, square pie charts take each percentage out of a total 100%.

#### References

- https://en.wikipedia.org/wiki/Pie_chart
- https://bl.ocks.org/bill-kidwell/2d09c9892747495592a7eb009d4d238d 3D Donut (d3.js v4)
- https://naver.github.io/billboard.js/demo/#Chart.PieChart
- https://naver.github.io/billboard.js/demo/#Chart.DonutChart
- http://d3pie.org/

### 6.7.4 Bullet Graph

A *bullet graph* is a variation of a bar graph developed by Stephen Few. Seemingly inspired by the traditional thermometer charts and progress bars found in many dashboards, the bullet graph serves as a replacement for dashboard gauges and meters. Bullet graphs were developed to overcome the fundamental issues of gauges and meters: they typically display too little information, require too much space, and are cluttered with useless and distracting decoration. The bullet graph features a single, primary measure (for example, current year-to-date revenue), compares that measure to one or more other measures to enrich its meaning (for example, compared to a target), and displays it in the context of qualitative ranges of performance, such as poor, satisfactory, and good. The qualitative ranges are displayed as varying intensities of a single hue to make them discernible by those who are color blind and to restrict the use of colors on the dashboard to a minimum.

#### References

- https://en.wikipedia.org/wiki/Bullet_graph
- https://github.com/GordonSmith/d3-bullet

### 6.7.5 Isotype

*Isotype* (International System Of Typographic Picture Education) is a method of showing social, technological, biological and historical connections in pictorial form. It consisted of a set of standardized and abstracted pictorial symbols to represent social-scientific data with specific guidelines on how to combine the identical figures using serial repetition.

#### References

- https://en.wikipedia.org/wiki/Isotype_(picture_language)
- http://bl.ocks.org/alansmithy/d832fc03f6e6a91e99f4 Pictogram grid in d3js
- https://bl.ocks.org/lelandlee/da2312cfbfcb5a311e68 Isotype - Squared
- https://bl.ocks.org/lelandlee/e50859751f3b096e3b27 Isotype Donut Art?
- http://bl.ocks.org/alandunning/51c76ec99c3ffee2fde6923ac14a4dd4 Bubble Matrix Chart V4

## 6.8 Time-series Visualizations

### 6.8.1 Line Chart

A *line chart* or *line graph* is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. It is similar to a scatter plot except that the measurement points are ordered (typically by their x-axis value) and joined with straight line segments. A line chart is often used to visualize a trend in data over intervals of time – a time series – thus the line is often drawn chronologically. In these cases they are known as run charts.

#### References

- https://en.wikipedia.org/wiki/Line_chart
- https://naver.github.io/billboard.js/demo/#Chart.TimeseriesChart
- https://naver.github.io/billboard.js/demo/#Chart.SplineChart
- https://bl.ocks.org/d3noob/ced1b9b18bd8192d2c898884033b5529 v4 curve interpolation comparison
- http://bl.ocks.org/emmasaunders/c25a147970def2b02d8c7c2719dc7502 Interpolation (v4)

### 6.8.2 Area Chart

An *area chart* or *area graph* displays graphically quantitative data. It is based on the line chart. The area between axis and line are commonly emphasized with colors, textures and hatchings. Commonly one compares with an area chart two or more quantities.

Area charts which use vertical and horizontal lines to connect the data points in a series forming a step-like progression are called step-area charts.

Area charts in which data points are connected by smooth curves instead of straight lines are called *spline-area charts*.

#### References

- https://en.wikipedia.org/wiki/Area_chart
- https://naver.github.io/billboard.js/demo/#Chart.AreaChart
- https://naver.github.io/billboard.js/demo/#Chart.StackedAreaChart

### 6.8.3 Radar Chart

A *radar chart* is a graphical method of displaying multivariate data in the form of a two-dimensional chart of three or more quantitative variables represented on axes starting from the same point. The relative position and angle of the axes is typically uninformative.

The *radar chart* is also known as *web chart*, *spider chart*, *star chart*, *star plot*, *cobweb chart*, *irregular polygon*, *polar chart*, or *Kiviat diagram*. It is equivalent to a parallel coordinates plot in polar coordinates.

#### References

- https://en.wikipedia.org/wiki/Radar_chart
- http://bl.ocks.org/nbremer/6506614 D3.js - Radar Chart or Spider Chart

### 6.8.4 Bar Chart

A *bar chart* or *bar graph* is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value. Some bar graphs present bars clustered in groups of more than one, showing the values of more than one measured variable.

Bar graphs can also be used for more complex comparisons of data with grouped bar charts and stacked bar charts. In a *grouped bar chart*, for each categorical group there are two or more bars. These bars are color-coded to represent a particular grouping. For example, a business owner with two stores might make a grouped bar chart with different colored bars to represent each store: the horizontal axis would show the months of the year and the vertical axis would show the revenue. Alternatively, a *stacked bar chart* could be used. The stacked bar chart stacks bars that represent different groups on top of each other. The height of the resulting bar shows the combined result of the groups. However, stacked bar charts are not suited to datasets where some groups have negative values. In such cases, grouped bar chart are preferable.

#### References

- https://en.wikipedia.org/wiki/Bar_chart
- https://naver.github.io/billboard.js/demo/#Chart.BarChart
- https://naver.github.io/billboard.js/demo/#Chart.StackedBarChart

### 6.8.5 Radial Bar Chart

#### References

- http://bl.ocks.org/kgryte/5926740 Nightingale's Rose + D3.js

### 6.8.6 Calendar Heat Map

A *heat map* (or *heatmap*) is a graphical representation of data where the individual values contained in a matrix are represented as colors. The term 'heat map' was originally coined and trademarked by software designer Cormac Kinney in 1991, to describe a 2D display depicting financial market information, though similar plots such as shading matrices have existed for over a century.

#### References

- https://bl.ocks.org/alansmithy/6fd2625d3ba2b6c9ad48 Heatmap Calendar in d3js
- https://github.com/wa0x6e/cal-heatmap
- http://prcweb.co.uk/lab/energy/ Visualisation of domestic energy consumption

## 6.9 Temporal Visualizations

### 6.9.1 Timeline

- http://bl.ocks.org/denisemauldin/e6da337734f855c2a89666afb11dc329 d3.layout.timeline categorized timelines