Distributed installation of Upsource without using Docker

Hardware requirements

Hardware requirements can be found here:
https://www.jetbrains.com/help/upsource/upsource-cluster-hardware-requirements.html

Setting up an Upsource cluster

This is a step-by-step instruction on how to install Upsource in a distributed multi-node cluster. This installation procedure should only be performed once.

The moving parts

An Upsource cluster consists of the services listed below. The services can either be installed on the same server or distributed across multiple servers – physical or virtual – in any combination. For some services only one instance is allowed, while others (*frontend*, *psi-agent*, *analyzer*) can be scaled as necessary. We recommend running scaled services on different servers to improve performance and reliability.

frontend	Provides the Upsource web-based UI.
psi-agent	Provides code model services (code intelligence) based on IntelliJ IDEA.
psi-broker	Manages *psi-agent* tasks.
analyzer	Imports revisions from VCS to Cassandra database.
opscenter	Provides cluster monitoring facilities.
haproxy	Provides the entry point of a distributed Upsource cluster. Proxies incoming requests to services.

file-clustering	Provides the backend for certain “smart” features of Upsource (review suggestions, revision suggestions, etc.) by computing file and revision similarity indices.

Additionally, Upsource depends on two external services: **JetBrains Hub**, which is used for user authentication and permissions management, and **Apache Cassandra**, which is the database engine used by Upsource. Cassandra itself can run both in single- and multi-node configurations, however, administration of Cassandra is beyond the scope of this document.

Prerequisites

Before installing the Upsource services you need to install the external ones: Apache Cassandra 3.10 and JetBrains Hub.

Installing Hub

Follow this instruction to install Hub:
https://www.jetbrains.com/help/hub/Install-and-Configure-Hub.html

Installing Cassandra

Please consult the Cassandra documentation for instructions on deploying Cassandra. Note that the following additional requirements are in place:

1. Cassandra 3.10 should be used.
2. Additional libraries should be added to the Cassandra installation (typically under `<cassandra_home>/libs`). The libraries for the specific build of Upsource can be downloaded from:

   http://download.jetbrains.com/upsource/cassandra-deploy-libs-{upsource_version}.zip

   e.g. http://download.jetbrains.com/upsource/cassandra-deploy-libs-2017.2.2197.zip

3. The following properties should be adjusted in cassandra.yaml file as follows:

```
batch_size_warn_threshold_in_kb: 250
batch_size_fail_threshold_in_kb: 5000
compaction_throughput_mb_per_sec: 32
```

Here you can find the basic instruction for a single-node Cassandra configuration.

Please note, that Apache Cassandra is the only database engine Upsource can use: the nature of the data stored and manipulated by Upsource precludes the use of typical SQL databases.

Configuring an Upsource cluster

Unpack upsource-services.zip:

http://download.jetbrains.com/upsource/upsource-cluster-services-<upsource_version>.zip
e.g. http://download.jetbrains.com/upsource/upsource-cluster-services-2017.2.2197.zip

The ZIP distribution contains the seven services described above as well as two files with environment variables that will be described below:

- `upsource.common.env`
- `service.specific.env`

`upsource.common.env` file contains the common properties that should be identical for all services:

```
############### upsource.common.env ################
CASSANDRA_HOSTS=cassandra_hosts

CASSANDRA_NATIVE_TRANSPORT_PORT=9042

// Will depend on the number of nodes
UPSOURCE_CASSANDRA_REPLICATION_FACTOR=1

// Set to “false” for multi-node
UPSOURCE_CASSANDRA_SINGLE_NODE=true

UPSOURCE_DATABASE=datastax

HUB_URL=public_hub_url

// If not provided, HUB_URL will be used
HUB_URL_INTERNAL=internal_hub_url

// ID of Upsource service in Hub
UPSOURCE_SERVICE_ID=service_id

// Secret of Upsource service in Hub
UPSOURCE_SERVICE_SECRET=service_secret
```

```
// The URL used by the end users to access Upsource
UPSOURCE_URL=upsource_url

// The port the Upsource web server will listen on, 8080 by default
UPSOURCE_EXPOSED_PROXY_PORT=port

UPSOURCE_HUB_CHECK_INTERVAL_SECONDS=600

// Number of threads used for initializing new projects
UPSOURCE_ANALYZER_THREADS_INIT_CLUSTER=2

UPSOURCE_MONITORING_LISTEN_PORT=monitoring_port

UPSOURCE_PSI_BROKER_HOST=psi_broker_host

UPSOURCE_PSI_BROKER_LISTEN_PORT=psi_port

// The path where Upsource backups will be stored (on the opscenter host)
// <upsource_opscenter_home>/backup is used by default
BUNDLE_BACKUP_LOCATION=backup_location

// Usage is described here
HUB_KEYSTORE_PATH=

// Usage is described here
HUB_KEYSTORE_PASSWORD=

// Host where upsource-opscenter is running
MONITORING_HOST=

// Location of haproxy.cfg, by default at /usr/local/etc/haproxy
HAPROXY_CONF_LOCATION=

// By default at /opt/upsource-haproxy
HAPROXY_SCRIPTS_LOCATION=
############### upsource.common.env ################
```

Properties should be exported as environment variables on each machine where services are running. For example, with the following command:

```
set -o allexport; source /path/to/file/upsource.common.env; set +o allexport
```

Note: Logs are located at `<upsource_service_home>/logs`. We recommend to use symlinks to customize this path.

`service.specific.env` contains individual properties that will be different for each specific service:

```
############### service.specific.env ################
UPSOURCE_SERVICE_MESSAGING_PORT=

UPSOURCE_FRONTEND_PORT=

// Leave it blank for singleton services
UPSOURCE_SERVICE_INSTANCE_ID=

// Temporary files. By default at <upsource_service_home>/tmp
UPSOURCE_TEMP_LOCATION=

// Application data. By default at <upsource_service_home>/data
UPSOURCE_DATA_LOCATION=
############### service.specific.env ################
```

To run a service with an individual properties file you can use the following syntax:

```
(set -a; . /path/to/service.specific.env; set +a;
/path/to/upsource-<service>/bin/upsource-<service>.sh start)
```

Note: the user account used to run Upsource services should have read/write permissions to all directories that are listed in both configuration files.

Starting an Upsource cluster

Before starting the Upsource services make sure that Cassandra and JetBrains Hub are running. Having done that, run the *upsource-cluster-init* service. It will prepare the required keyspaces in Cassandra and exit.

The remaining Upsource services can be launched in any order using the following command:

```
(set -a; . /path/to/service.specific.env; set +a;
/path/to/upsource-<service>/bin/upsource-<service>.sh start)
```

The status of all services can be checked on the monitoring page at *<upsource_url>/monitoring*.

Scaling Upsource services in the cluster

The following services can be scaled:

- frontend
- psi-agent
- analyzer

Scaling the frontend service

Frontend service should be scaled in installations with a large number of users as well as to improve availability. We suggest doing that using a combination of *haproxy* and Python scripts that are described below.

Before configuring haproxy and load balancer make sure that the following packages are installed on the server:

- haproxy 1.6.7
- python 2.7
- python-pip
- jinja2

The load balancer can be downloaded here and consists of the following scripts and configs:

- `run.sh`
- `reloader.sh`
- `loadbalancer.py`
- `/conf/haproxy`
    - `haproxy.cfg.tmpl`
    - `haproxy_503.http.tmpl`
    - `haproxy_504.http.tmpl`
    - `initial.json`

Put the haproxy configs from /conf/haproxy to `${HAPROXY_CONF_LOCATION}/conf/haproxy`.

Put the scripts to `${HAPROXY_SCRIPTS_LOCATION}` (`/opt/upsource-haproxy` by default).

Launch `run.sh`

Note that the user account used to launch `run.sh` should have read/write permissions to the directories defined in `HAPROXY_CONF_LOCATION` and `HAPROXY_SCRIPTS_LOCATION`.

Scaling the analyzer

When dealing with extremely large and/or active projects (hundreds of thousands of commits overall, hundreds of daily commits) it may be necessary to set up a dedicated analyzer for processing them. The subset of projects the particular analyzer works on is defined by the environment variable `ANALYZER_PROJECTS`. Its value is specified using a mask where the following symbols have a special meaning:

`+:` stands for “include”
`-:` stands for “exclude”
`.+` stands for “all projects”

For example:

`ANALYZER_PROJECTS=+:.+` means “process all projects”

`ANALYZER_PROJECTS=-:.+,+:Project-A` means “exclude all projects, include Project-A” (process Project-A only)

`ANALYZER_PROJECTS=+:.+,-:Project-A` means “include all projects, exclude Project-A” (process everything but Project-A)

`ANALYZER_PROJECTS=-:.+,+:Project-A,+:Project-B` means “exclude all projects, include Project-A, include Project-B” (process Project-A and Project-B)

When a dedicated analyzer is set up for a specific project, other analyzers should be instructed to exclude this project from processing: two analyzers can't work on the same project.

Scaling the psi-agent

The same logic applies to scaling the psi-agent. The list of projects is defined by the `UPSOURCE_PSI_PROJECTS` environment variable.