

Werkzeuge und Infrastrukturen

Wintersemester 2016/17

Ahmed Albawabiji

Professur für Korpuslinguistik
Studiengang Linguistische Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg
ahmed.albawabiji@fau.de

Erlangen, 20.12.2016



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

PHILOSOPHISCHE FAKULTÄT
UND FACHBEREICH THEOLOGIE

Inhaltsverzeichnis

- 1 Einführung
 - Pipelines

- 2 Hauptteil
 - Luigi
 - Airflow
 - Pinball

- 3 Zusammenfassung und Vergleich

- 4 Literaturverzeichnis

Abstakt: Was sind Pipelines?

Es ist die Aktion von Serialisierung.

Eine Gegenaktion ist die Deserialisierung oder Parallelisation.

Es ist eine Methode, die Ausgabe eines Prozesses mit der Ausgabe eines anderen Prozesses zu verbinden.

Motivation

Früher sah es so aus:

```
$ pythonget_some_data.py  
$ pythonclean_some_data.py  
$ pythonjoin_other_data.py  
$ pythondo_stuff_with_data.py
```

funktioniert gut und sogar schnell!

Problem beim Weiterleiten!

Programmierer(B):

Es hat nicht funktioniert...!?

Programmierer (A):

Hast du den die Daten schon gefiltert, bevor du sie
miteinanderverbinden!?

Eine mögliche Lösung:

```
do_everything.pyscript:  
if __name__ == '__main__':
```

**Darunter schreibe ich alle zum Laufen gewollten Methoden
Und dann führe ich alles mit einem Befehl aus!**

python do_everything.py

Warum Pipelines?

- Unsere Daten leicht bearbeiten und die Fehler besser auflösen.
- Unsere Arbeit ganz ordentlich organisieren.
- Mehr Kollaborationsmoeglichkeit, besser Codeverständnis zwischen den Programmierern und/oder den verschiedenen Betriebssystemen oder Editors. (Das Abschicken ist nicht immer sicher!).
- Daten von irgendwo holen, etwas damit machen und dann schicken sie wo anderes ab.

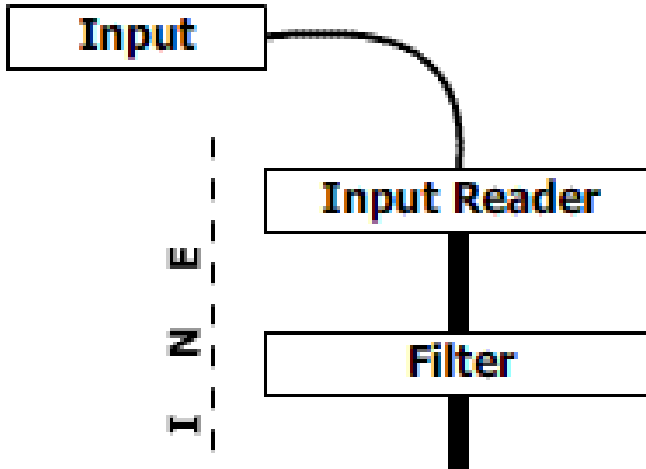
Was sind Pipelines?

- Es ist eine Methode, die die Ausgabe eines Prozesses mit der Ausgabe eines anderen Prozesses miteinander verbindet.
- Pipes sind alle wichtigen Schritte oder Prozesse zur Vorbereitung oder /und Verarbeitung unsere Daten.
- Es ist die Umformen von prototype zur Produktion.
- So wird die Ausgabe einesProgramms als die Eingabe eines anderen Programm verwendet um am Ende etwas komplizierteres draus herauszubringen.
- Data pipelines sind überall.
- Nützlich von data als Ereignis(Produktion) zu denken!
- Wir schreiben Programme die zusammen ein grosseres Programm machen.

Bestandteile einer Pipe:

- Standard input(stdin) : Datei, von der wir Information bekommen, um damit bearbeiten zu können.
- Standard output(stdout) : Datei, von der wir Information bekommen, um damit bearbeiten zu können.
- Standard error(stderr) : !! OOPS! Etwas ging schief! Bitte schön diese Fehlermeldung als Ausgabe ! stdout(1) processes (A) verbinden wir mit dem stdin(0) Prozesses (B) durch den Bar | (ist ja trivial)
Command(A) | Command (B)
Deswegen sind sie auch FIFO genannt(first in first out)

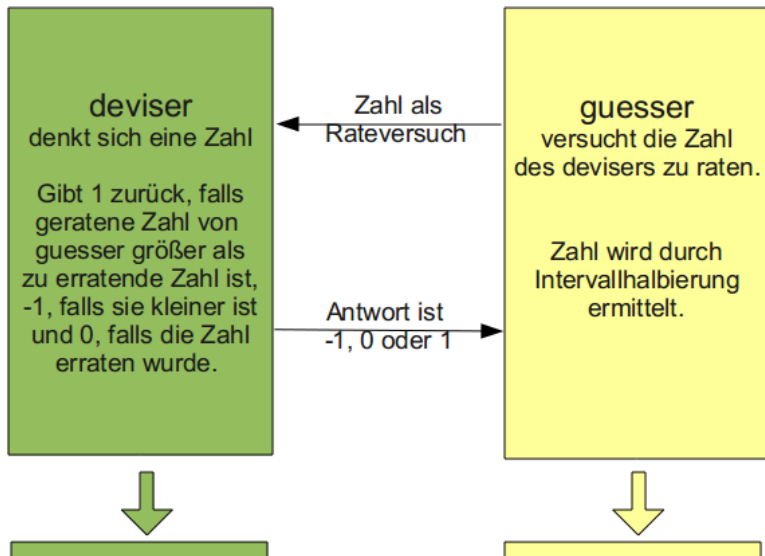
A Simple Pipeline



Arten von Pipes



Arten von Pipes





kleine Motivation

Führen wir zurück zu unserer Motivation von Pipelines:

Aber was wenn es Fehler dran gäben, was macht man!

Man könnte das mit (try : und except) lösen!

Danach werden wir einen Weihnachtsbaum haben! Umständlich!

Da könnten uns Packages wie Luigi, Airflow, Pinball oder auch andere retten.

<https://github.com/pditommaso/awesome-pipeline>

Was ist Luigi

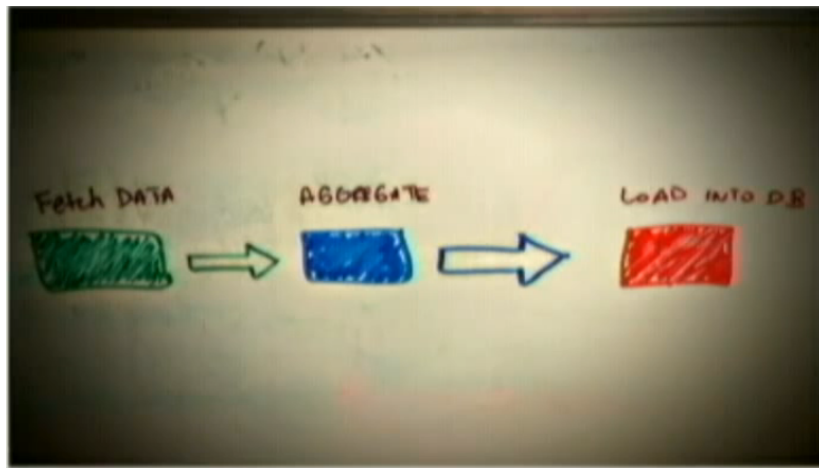
- Luigi ist ein python package, das dabei hilft kompezierte Pipelines zu bilden.
- Wurde vom Spotify entwickelt, um ihnen thousande von Jobs, die sie täglich in einer bestimmten Reihenfolge führen, ermöglicht zu bewältigen.
- Es ist der Klebstoff, der die Prozesse miteinander zusammenbringt.
- Abhängigkeit ist durch input und Output identifiziert. D.h. TaskB ist von TaskA abhängig .. bedeutet, Output von TaskA ist das Input (requires) von TaskB.

Vorteile von Luigi

- Definieren von einander abhängige Tasks, die lange Zeit zum Laufen brauchen.
- Tasks verketteten, automatisieren, mit den daraus möglichen entstandenen Fehlern umgehen, die Reihenfolge sehr einfach ändern. Tasks könnten alles sein, Daten von Datenbanken holen oder zu den Datenbanken schicken, maschinale lernen oder was anderes.
- weiterzumachen von dem letzten laufenden Schritt.(muss nicht nochmal von Anfang)
- Workflow management, Visualization(running, pending, done Tasks!)(dafür braucht man \$ luigid). Aber:
Ausführung ist von der Reihenfolge abhängig.
keine vorhandenen Reihenfolgen!
Visualisierung könnte besser sein!

\$pip install luigi

Magie von Luigi



Drei Hauptfunktionen:

Task: Ein Teil der Bearbeitung! Erweiterung von der Klasse `luigi.Task`.

Target: Output eines Tasks..kann alles sein! Datei erstellen, modifizieren oder sonstiges!

- `def requires(self):`
 `pass # list of dependencies`
- `def output(self):`
 `pass # task output`
- `def run(self):`
 `pass # tasks logik.`

Für ein Task sind die drei erforderlich . . . wir könnten aber unsere Kette vergrößern.

Beispiele dazu..

```
import luigi

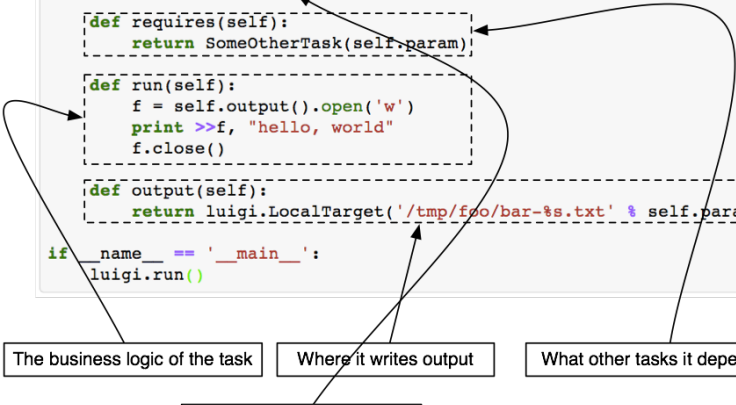
class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```



The business logic of the task

Where it writes output

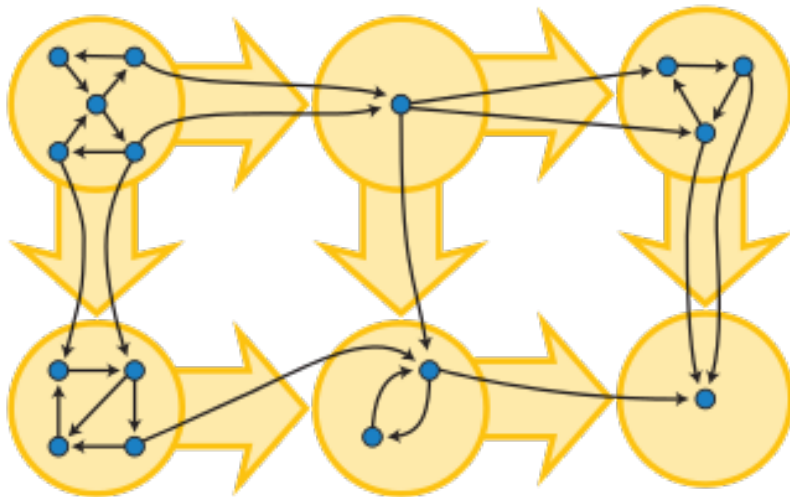
What other tasks it depends on

Parameters for this task

Airflow

- Funktioniert mit Python, my SQL, Bash und auch anderen.
- Seit 2015.
- DAGs anstatt Pipelines. Directed Acyclic Graph: Eine Sammlung von den Tasks, die wir zum Laufen bringen wollen! In einer Art und Weise, die die Relationen und Abhängigkeit zwischen denen spiegelt.
DAGs werden in einer Datei in Airflow's Datei, die *DAG_FOLDER* heisst.
- Beim ersten Lauf erstellt das Programm `airflow.cfg` und da kann man Airflow konfigurieren. Beispiel Task A sammelt Daten , B analysiert , C schickt Emails.
`$pip install airflow`
Strukturierter Baum:
<https://airflow.incubator.apache.org/ui.html>

Ein DAG



- Auch Python Package.
- Modifizierter PunktWorkflow manager and scheduler.
- Werden parent-job und child-Job benutzt, um die Abhängigkeit zu bilden.
- Meldungen durch die E-Mail. \$ pip install pinball

Luigi Vs airflow Vs pinball

[https://www.michaelcho.me/article/
data-pipelines-airflow-vs-pinball-vs-luigi](https://www.michaelcho.me/article/data-pipelines-airflow-vs-pinball-vs-luigi)

Wachsmuth, Henning: Text Analyse und Pipelines, Springer, 2015.

<https://github.com/pditommaso/awesome-pipeline>

<http://bionics.it/posts/luigi-tutorial>

[https://www.michaelcho.me/article/
data-pipelines-airflow-vs-pinball-vs-luigi](https://www.michaelcho.me/article/data-pipelines-airflow-vs-pinball-vs-luigi)

<http://www.python-kurs.eu/pipes.php>