Kirby

Adimian

Sep 23, 2019

CONTENTS

CHAPTER ONE

KIRBY

Kafka-based Stream Processing Framework

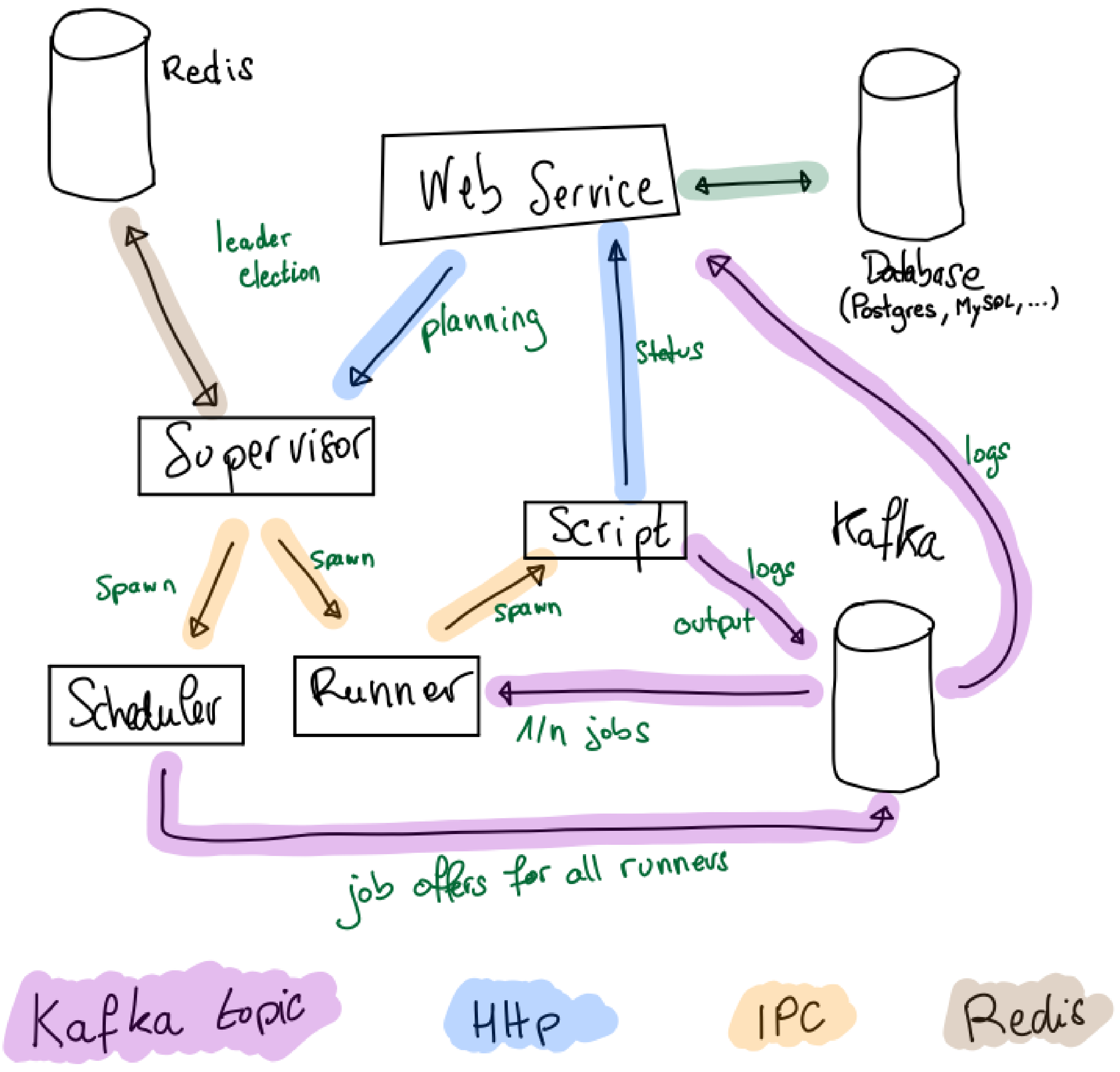
Please see the official documentation here: http://kirby.readthedocs.io

CHAPTER

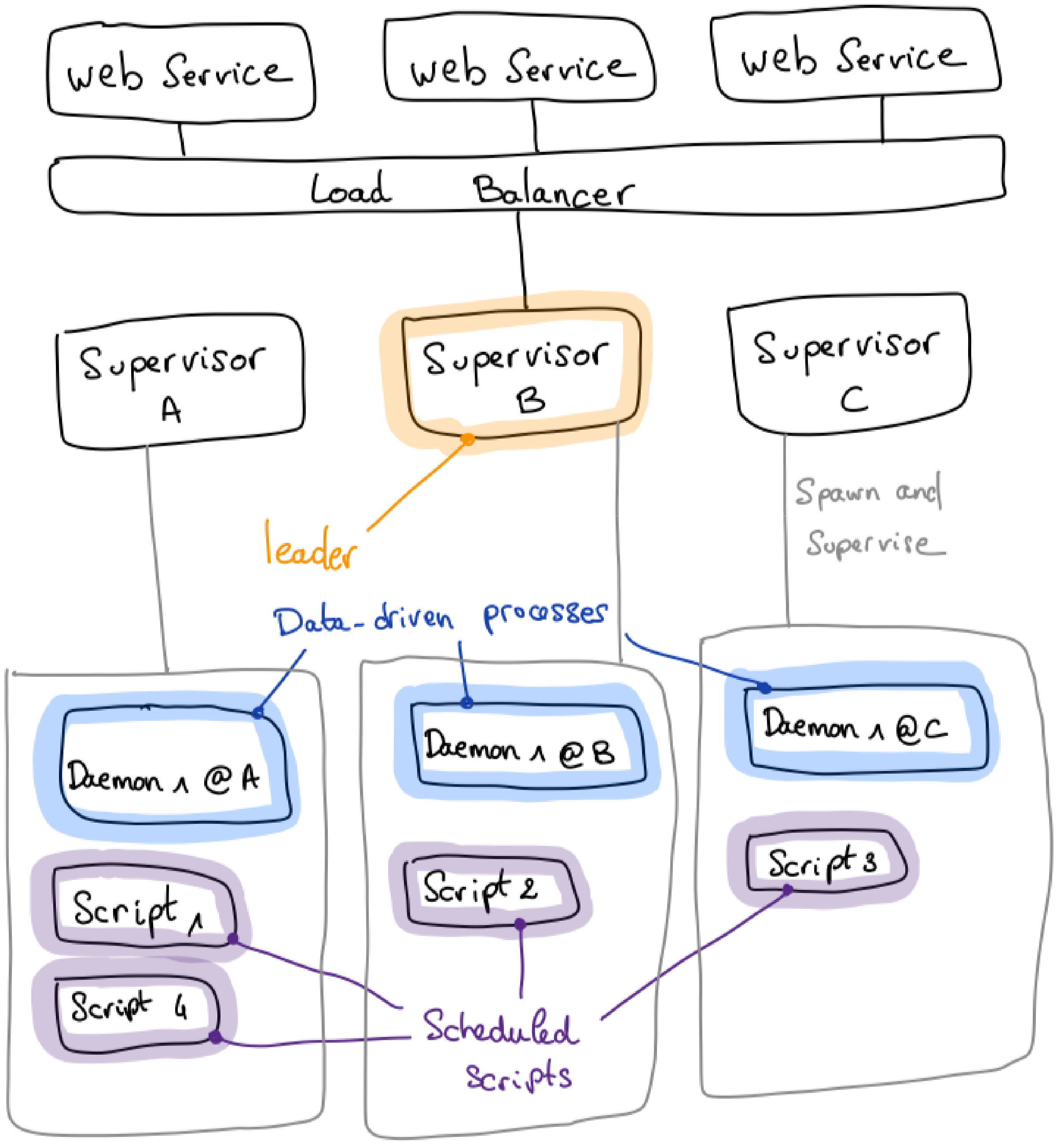
TWO

COMPONENTS

2.1 Services



2.2 Processes



CHAPTER THREE

FEATURES

3.1 For Infrastructure/Ops

3.1.1 De-centralized worker deployment

- Scripts are deployed from the web UI
- Versioned, can co-exist in many versions at once
- If you can publish to PyPi, it can be deployed

3.1.2 Real-time monitoring

- Jobs send their status (success / fail) and logs in real time
- Notifications can be configured per job to be triggered in case of failure or missed mark (should be running but is not)

3.1.3 Centralized secrets management

- Scripts pull their configuration from the "vault"
- Secrets are stored in a central place
- Secrets are never persisted on disk
- ACL apply to secret access

3.1.4 Multiple authentication backends (local, ldap, . . .)

- Authentication
  - LDAP
  - Okta
  - Local accounts
- Authorization
  - Groups
  - Roles

3.2 For Developers

3.2.1 Workers as simple python scripts

- Scripts are simple Python scripts
- You can use whatever library you want, as long as it can be pip-installed
- No hard requirement on using Kirby at all
- Configuration is passed through env variables

3.2.2 File-based local unit-testing

- Kirby is meant to be testable
- All input / output in Kirby is a message
- Messages are simulated using folders and files
- Comes with integrated testing helpers

3.2.3 Simple integration to web frameworks

- Designed to allow swap-in replacement of Celery
  - ex: produce data in an ETL script, use it straight in your app, or the other way around
- Chord / group logic is replaced by stream piping
- Higher throughput and parallelism
- Simpler status management and failure recovery
- Not limited to running jobs, integrations can also serve as a view on all Kirby-managed data
- Can be used in templates, views, etc.

3.3 For Data Engineers

3.3.1 Scheduled tasks with exceptions

- Schedule can be configured with cron-like syntax
- Add exceptions to pause or slow down a job from running (when source is broken for instance)
- Automatically back to normal schedule at the end of the exception, no need to undo them

3.3.2 Rewind/replay of tasks

- Rewind a data source and all downstream processing will flow naturally (but will flag the data as the result of a re-run in case it matters)
- Replay production data in preprod or even on a developer laptop $\rightarrow$ testing with prod data = easier development / debugging

3.3.3 Data traceability with data dependency graph documentation

- Each process adds itself in the message headers, to allow end-to-end data traceability
- Visualize data flows in the web UI to understand
  - where your data comes from
  - when it was processed
  - what transformations were applied

3.3.4 Computed tables

- Kirby **State** objects allows you to keep a persistent object that can be updated by scripts
- By default, a state only shows its latest version, but each mutation is versioned and can be recovered (only limited by disk space)
- Useful for "hot" data with expensive computation costs

CHAPTER

FOUR

USER GUIDE

In order to use Kirby, you will need several services running. The web interface is the only component that needs access to the database.

4.1 Kirby managed services

- the `kirby` web UI, to configure your Kirby cluster
- **at least one** `kirby` supervisor to trigger jobs and execute them

Note: We recommend using your operating system process supervisor such as *systemd* to run *kirby*.

Todo: provide systemd templates

4.2 Supporting services

- a Kafka cluster (required by all `kirby` services)
- a database server (required for the `kirby` web UI)

Note: Running both services on the same machine is fine for development

4.3 Installing Kirby

```
$ pip install -U kirby
```

4.4 Running the web interface

```
$ kirby web [--host 127.0.0.1] [--port 8080]
```

Important: We recommend you not to expose the web service directly on the Internet, but to use a reverse proxy such as Nginx or HAProxy

4.4.1 Adding a superuser

If you want to add a local user (so not using external user provisioning like LDAP or Okta), you can use the following command on the web UI server

```
$ kirby demo
demo data inserted in the database
```

4.4.2 Demonstration database

If you just want a quick preview of Kirby's features, you can summon the demo database as follows:

Warning: Please only use on an empty database, it will mess with your existing data and there is no rollback mechanism.

```
$ kirby adduser alice
Password: ******
Give admin rights? [y/N]: y
User alice added with admin rights
```

4.5 Running the supervisor

Important:

- There must be at least one *supervisor* instance running at all times.
- Each supervisor must have a unique name

If you want to call your instance "server-1" then start kirby as follows:

```
$ kirby supervisor server-1 [--window 5] [--wakeup 30]
```

- `window` is the frequency at which the supervisor tries to elect itself as the cluster leader. Use longer interval if your network is too noisy and you do not have scheduled jobs.
- `wakeup` is the shortest interval between two scheduled jobs. Use longer interval if your network is limited and you do not have scheduled jobs or if the intervals are very long.

Note: Defaults are fine in most cases.

CHAPTER FIVE

CONTRIBUTION GUIDELINES

5.1 Developer Workstation Setup

1. install Kafka https://kafka.apache.org/quickstart
2. install Redis https://redis.io/topics/quickstart
3. clone the kirby repository `git clone git@github.com:adimian/kirby.git`

5.2 Useful Links

- Source: http://github.com/adimian/kirby
- Issues: http://github.com/adimian/kirby/issues
- Documentation: http://kirby.readthedocs.io

5.3 Writing Documentation

Todo: write this

CHAPTER

SIX

COMPARISON WITH OTHER PLATFORMS

Kirby is still in active development, and aims to become a platform for data processing. It draws its inspirations from many existing products. Some of them are listed here to help you evaluate whether Kirby is the right choice for you.

Important: Kirby developers are not specialists in the following products, our opinion is expressed here in good faith but could be inaccurate. Corrections and additions are welcome, to come up with the most helpful vision for potential users.

6.1 Airflow

https://airflow.apache.org

Airflow is a Python framework for building workflows. It is a very mature and robust platform, has lot of plugins for various tasks, and before considering using Kirby, you should first evaluate Airflow.

6.1.1 Similarities

- Airflow allows you to define jobs and schedules
- You can define tasks dependencies / successors
- It shows job status (pending / success / failed)
- It allows replay / rewind (called "backfill")

6.1.2 Differences

- Airflow is an execution framework, not a data-driven framework
- Scripts need to be manually copied on the server
- Metadata goes into a single database that can grow big fast
- Scripts only run in AirFlow, data storage has to happen outside of it
- DAGs are monoliths: you cannot tap into an existing flow, you either need to
  - coordinate and create your own DAG
  - modify the original DAG to add your own step

6.2 Faust

https://github.com/robinhood/faust

Faust is used to build data-driven pipelines using `asyncio`, porting the notion of Kafka Streams to Python

6.2.1 Similarities

- It has the concept of `Table` which is the inspiration for Kirby's `State`
- Supports deploying redundant workers and keeping them alive

6.2.2 Differences

- Built upon `asyncio`, which does not fit well with CPU-bound tasks
- No support for scheduled tasks, monitoring, . . .

6.3 Streamparse

https://github.com/Parsely/streamparse

6.3.1 Similarities

- Deploys packaged versions of your code on a cluster
- Supports data-driven tasks and scheduled tasks

6.3.2 Differences

- Requires an Apache Storm cluster
- Data storage is handled outside of Storm

6.4 Others

Todo: add more similar products

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search