Pipeline Architecture

February 26, 2024

Brae Webb & Richard Thomas

Presented for the Software Architecture course
at the University of Queensland

THE UNIVERSITY
OF QUEENSLAND
AUSTRALIA

Pipeline Architecture

Software Architecture

February 26, 2024

Brae Webb & Richard Thomas

1 Introduction

Pipeline architectures take the attribute of modularity of a system to the extreme. Pipeline architectures are composed of small well-designed components which can ideally be combined interchangeably. In a pipeline architecture, input is passed through a sequence of components until the desired output is reached. Almost every developer will have been exposed to software which implements this architecture. Some notable examples are bash, hadoop, some older compilers, and most functional programming languages.

Definition 1. Pipeline Architecture
Components connected in such a way that the output of one component is the input of another.

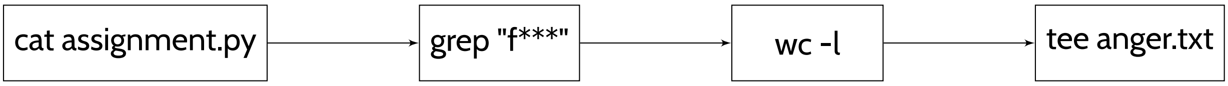

Figure 1: An example of using bash's pipeline architecture to perform statistical analysis.

The de-facto example of a well-implemented pipeline architecture is bash, we will explore the philosophy that supports the architecture shortly. The above diagram represents the bash command,

```
$ cat assignment.py | grep "f***" | wc -l | tee anger.txt
```

.

If you are unfamiliar with Unix processes (start learning quick!).

cat Send the contents of a file to the output.

grep Send all lines of the input matching a pattern to the output.

wc -l Send the number of lines in the input to the output.

tee Send the input to stdout and a file.

2 Terminology

As illustrated by Figure 2, a pipeline architecture consists of just two elements;

Filters modular software components, and

Pipes the transmission of data between filters.

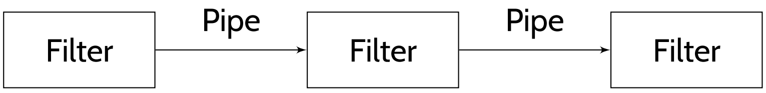

Figure 2: A generic pipeline architecture.

Filters themselves are composed of four major types:

Producers Filters from which data originates are called producers, or source filters.

Transformers Filters which manipulate input data and output to the outgoing pipe are called transformers.

Testers Filters which apply selection to input data, allowing only a subset of input data to progress to the outgoing pipe are called testers.

Consumers The final state of a pipeline architecture is a consumer filter, where data is eventually used.

The example in Figure 1 shows how bash's pipeline architecture can be used to manipulate data in Unix files. Figure 3 labels the bash command using the terminology of pipeline architectures.

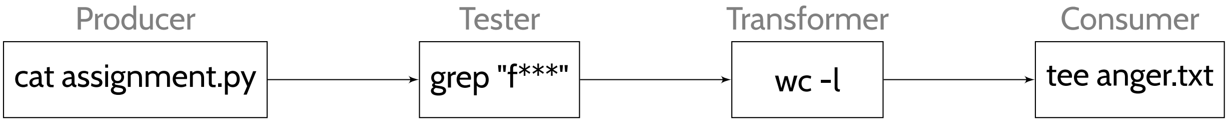

Figure 3: Figure 1 with labelled filter types.

3 Pipeline Principles

While the concept of a pipeline architecture is straightforward, there are some principles which should be maintained to produce a well-designed and re-usable architecture.

Definition 2. One Direction Principle

Data should flow in one direction, this is *downstream*.

The data in a pipeline architecture should all flow in the same direction. Pipelines should not have loops nor should filters pass data back to their *upstream* or input filter. The data flow *is* allowed to split into multiple paths. For example, figure 4 demonstrates a potential architecture of a software which processes the stream of user activity on a website. The pipeline is split into a pipeline which aggregates activity on the current page and a pipeline which records the activity of this specific user.

The One Direction Principle makes the pipeline architecture a poor choice for applications which require interactivity, as the results are not propagated back to the input source. However, it is a good choice when you have data which needs processing with no need for interactive feedback.

Definition 3. Independent Filter Principle

Filters should not rely on specific upstream or downstream components.

In order to maintain the reusability offered by the pipeline architecture, it is important to remove dependencies between individual filters. Where possible, filters should be able to be moved freely. In the example architecture in figure 4, the EventCache component should be able to work fine without the TagTime component. Likewise, EventCache should be able to process data if the Anonymize filter is placed before it.

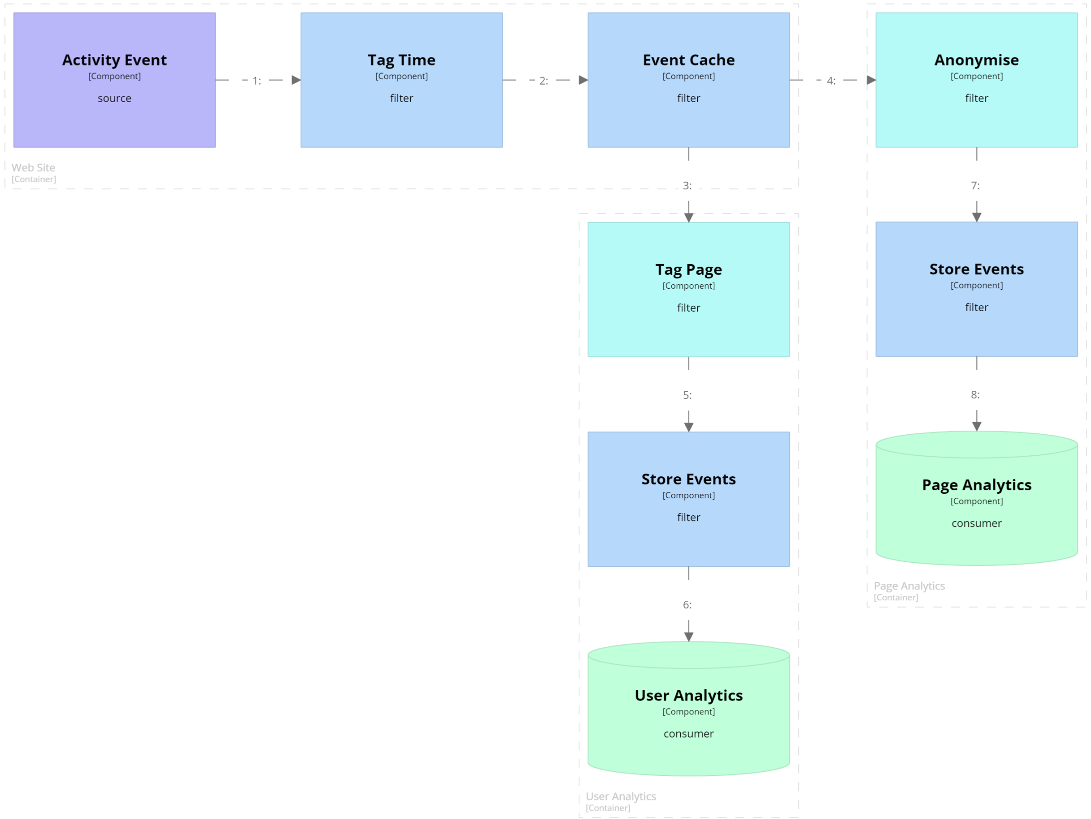

Figure 4: Pipeline architecture for processing activity on a website for later analytics.

Producers and consumers may assume that they have no upstream or downstream filters respectively. However, a producer should be indifferent to which downstream filter it feeds into. Likewise a consumer should not depend on the upstream filter.

Corollary 1. Generic Interface

The interface between filters should be generic.

Corollary 1 follows from definition 3. In order to reduce the dependence on specific filters, all filters of a system should implement a generic interface. For example, in bash, filters interface through the piping of raw text data. All Unix processes should support raw text input and output.

Corollary 2. Composable Filters

Filters (i.e. Transformers & Testers) can be applied in any order.

Corollary 2 also follows from definition 3. If filters are independent they can be linked together in any order. This means that a software designer can deliver different behaviour based on the order in which filters are linked. For example, in bash, applying a tester before or after a transformer can lead to different results.

4 Conclusion

A pipeline architecture is a good choice for data processing when interactivity is not a concern. Conceptually pipelines are very simple. Following the principles of a pipeline architecture will deliver a modular system which supports high reuse.