PipelineX: A Feature Animation Pipeline on Microservices

Dan Golembeski
DreamWorks Animation
dan.golembeski@dreamworks.com

Ray Forziati
DreamWorks Animation
ray.forziati@dreamworks.com

Ben George
DreamWorks Animation
ben.george@dreamworks.com

Doug Sherman
DreamWorks Animation
doug.sherman@dreamworks.com

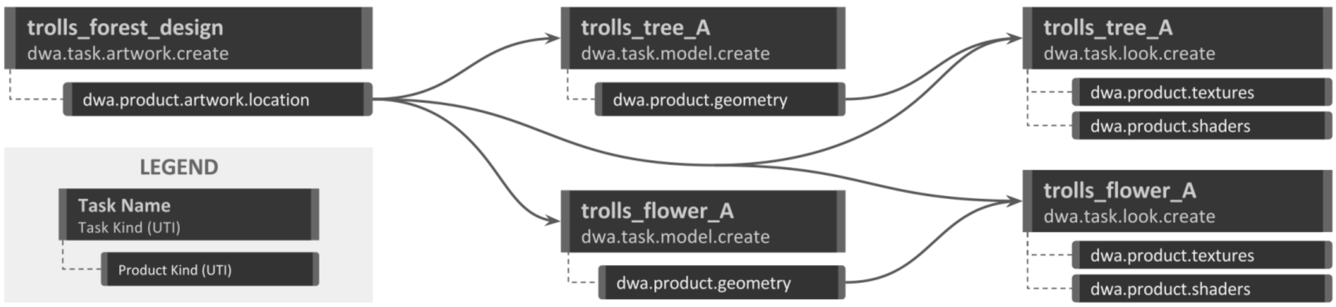

Figure 1: A simple example of a transactional workflow, an important piece of a service-based pipeline.

ABSTRACT

Here we present a unique approach to building a highly-scalable, multi-functional, and production-friendly feature animation pipeline on a core infrastructure comprised of microservices. We discuss basic service layer design as well as the benefits and challenges of moving decades-old production processes for an entire animation studio to a new, transactional pipeline operating against a compartmentalized technology stack. The goal is to clean up the clutter of a legacy pipeline and enable a more flexible production environment using modern, web-based technology.

CCS CONCEPTS

• **Networks** → **Cloud computing** • **Computing methodologies** → **Computer graphics** • *Computing methodologies* → *Graphics systems and interfaces* • General and reference → Design

KEYWORDS

feature animation, pipeline, microservices

ACM Reference format:

Dan Golembeski, Ray Forziati, Ben George, and Doug Sherman. 2017. PipelineX: A Feature Animation Pipeline on Microservices. In *DigiPro '17, Los Angeles, CA, USA, July 2017*, 4 pages.

https://doi.org/10.1145/3105692.3105702.

1 INTRODUCTION

Most feature animation production pipelines rely on a colorful set of scripts, plugins, databases, APIs, and naming conventions cobbled together over the course of many years. Historically, this approach has evolved from the need to simply glue together highly-varied workflows and technologies as quickly as possible, resulting in a pipeline that is time-consuming to debug, challenging to evolve, and hardwired to specific workflows and applications. However, after years of analyzing large-scale pipelines at various studios, we have concluded that this "pipeline problem" [Calude et al. 2014], while eternally difficult and complex, is exacerbated not only by the style of development but more so by the lack of a clearly delineated technology stack, comprised of distinctly separate layers of responsibility, optimally designed to contribute exactly what they each need to – and nothing more.

If we look outside the comforts of our industry, we realize that we share a common problem set with our colleagues in enterprise computing, such as social media, streaming entertainment, and online commerce. Those applications, at the core, are attempting to efficiently orchestrate large data sets among many distributed users. That is not far at all from our goal for a successful global production pipeline: making sure users (artists, in this case) have exactly what they need in order to do their work at the precise time they request it wherever they are.

Thus, our new approach to pipeline architecture is founded on the simple idea of similarly harnessing distributed resources through service-oriented architecture (SOA). In turn, this architecture enables us to leverage web-based and big data technologies that have already been validated and tested in enterprise computing. More specifically, we look to microservices, a flavor of SOA emphasizing independent distribution of smaller services, as a way to further support our efforts of compartmentalizing the pipeline.

2 HISTORICAL PRODUCTION USAGE

To date, a modest set of microservices is already being utilized on our feature animation productions for production data revision control, editorial cutlists, media identification, metadata searching, multi-site data transfer, and other use cases that represent specific, well-contained functionality within our production processes. In our implementations we've been able to leverage a wide variety of libraries, frameworks, languages, and databases so that each microservice can use the best technologies with the best approach to deliver successful production quality results. Java has been the language of choice for most production services, with an additional collection of complementary Python services. Choices for database technologies have included NoSQL solutions such as Couchbase, MongoDB, Cassandra, and Elasticsearch.

During production, we have observed demands on our existing services infrastructure peak around 110,000 transactions per second. In times of high demand like this, we've been able to apply a variety of caching and dynamic scaling techniques to maintain an acceptable level of performance for interactive and batch experiences. These achievements, with all the associated tweaks and configurations, have increased our confidence that a service-based infrastructure can indeed scale and flex to meet the ever-increasing demands of a modern feature animation production pipeline.

It is through all this prior testing and experience that we felt comfortable taking the next step to expand the suite of microservices to our entire production pipeline in an attempt to both solve the problems outlined in our introduction and yield new capabilities, flexibility, and scale that has not yet been achievable through more traditional approaches.

3 SERVICE ARCHITECTURE

In the following diagram (Fig. 2), we illustrate the types of services a typical feature animation pipeline would require. Note that these are merely descriptions for categories of services, not specific service implementations.

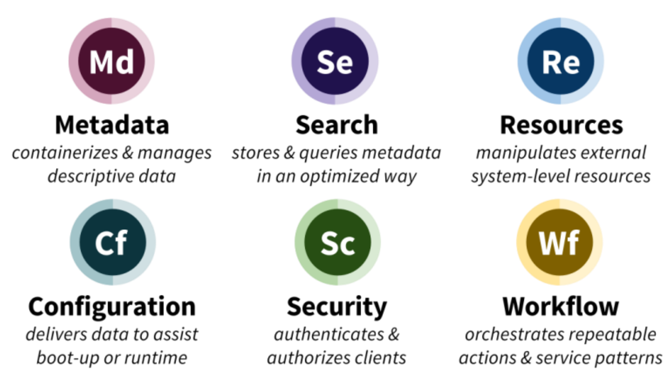

Figure 2: A clean separation of service responsibility.

In Fig. 3, we further illustrate a subset of the microservices implemented in our current architecture and how they interact in a common user workflow (in this case, creating a new asset). We use the same colored discs from Fig. 2 to indicate the type of service implemented in each step.

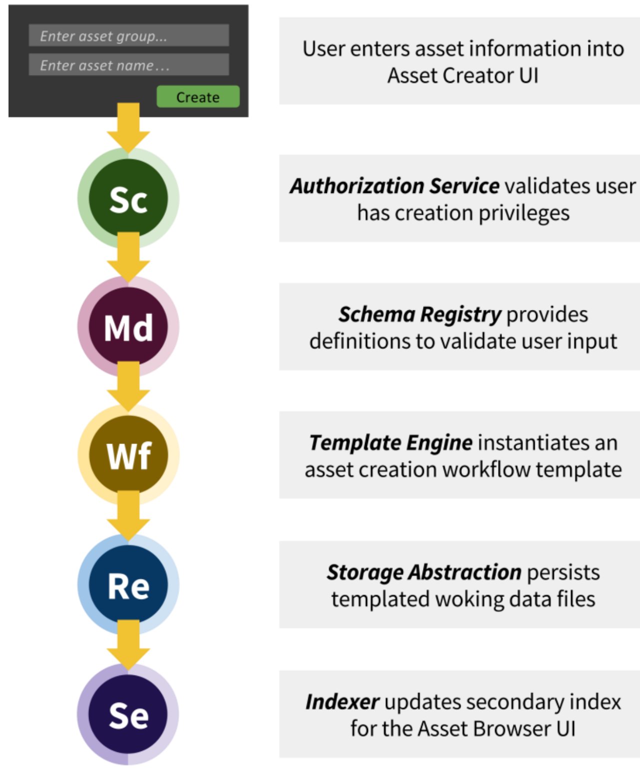

Figure 3: An example of services involved in asset creation.

To convey a more comprehensive scope of services in the PipelineX platform, we have included Fig. 4 on the following page. The figure outlines a minimal set of pipeline components:

- **Desktop Applications:** local applications communicating with backend services
- **Web Applications:** applications assisting in configuration, creation, management, and visualization aspects of pipeline data
- **Microservices:** delivers individual, fine-grained, lightweight, specific sets of functionality

These combined components build up to deliver an integrated user experience. This architecture allows for data to be optimally persisted, accessed, and presented in a way that meets the requirements of a production pipeline and most importantly the end user.

4 TRANSACTIONAL WORKFLOW

In addition to clean service architecture, building a pipeline on microservices requires a different model for how production work gets done. Many pipelines employ a *repository* model, in which hundreds of artists are simultaneously poking at the same

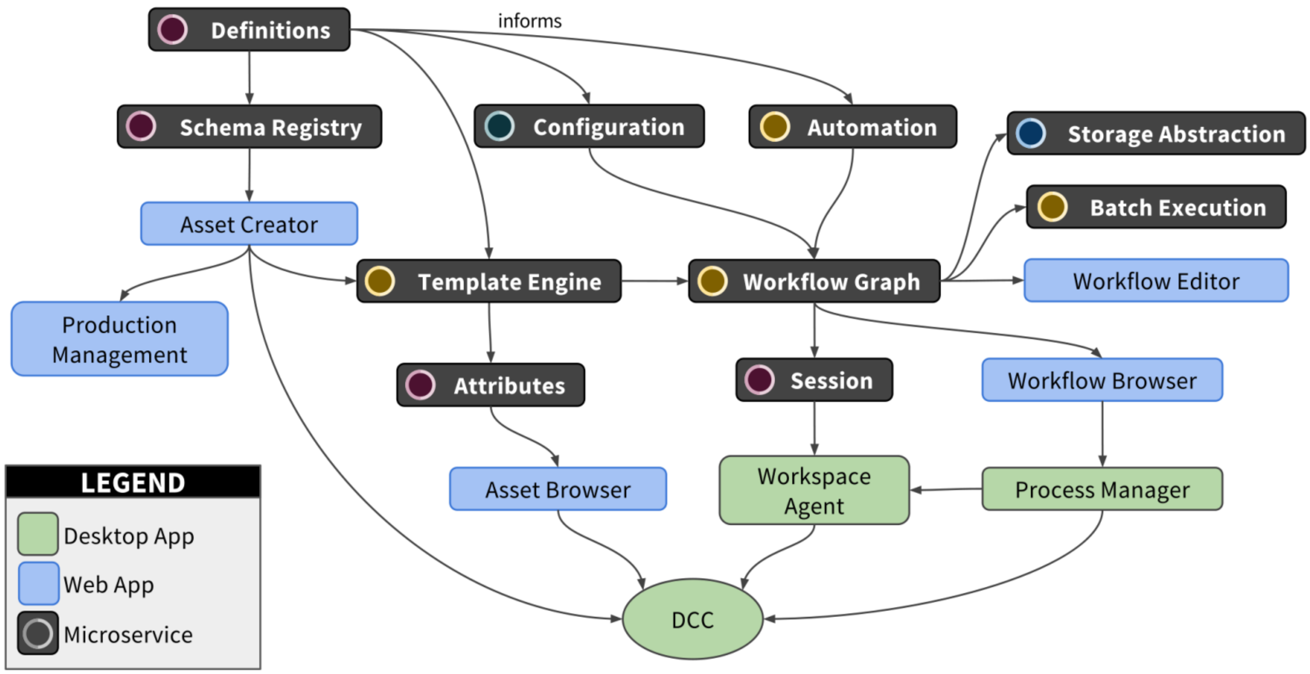

Figure 4: The conceptual components of the PipelineX architecture.

monolithic data sets, which is often bound by the bottlenecks of a centralized filesystem [Vanns et al. 2016]. Instead, we rely on a highly *transactional* model, in which each artist is assigned a scoped unit of work called a *task* (or set of tasks). To perform those tasks, an artist essentially imports datasets, then modifies or creates data, and finally exports new datasets. These datasets are commonly referred to as *products*. Downstream artists then pick up a subset of products, add to the data or further modify it, and send their results downstream as additional products. (See Fig. 1 for a simplified example of a task-product graph for basic asset creation workflows.) This results in a well-defined pipeline that is merely a large dependency graph of tasks and products modified by asynchronous and atomic operations, the exact types of operations that can be efficiently and easily processed by our microservice infrastructure.

When an artist wants to begin some work, we are able to populate clean and transient workspaces with production data as needed based on metadata from the pipeline service layer. Next, we give the artist access to the tools unique to his or her task - and then get out of the way so the artist can perform the work. When the artist is ready to complete an iteration, we register the work and effectively clean up the workspace. This process simply repeats wherever and whenever is necessary.

This design keeps the pipeline layer lightweight and out of the way. Its role can be contained to simply processing events triggered by workflow tools and only assists the users in starting or completing their tasks.

5 COMPARATIVE ANALYSIS

The concept of a transactional pipeline is not new to PipelineX. In fact, generally speaking, capturing pipeline transactions in a dependency graph is already a commonly accepted technique for tracking pipeline dataflow, even for pipelines not utilizing SOA [Johnson et al. 2014; Polson 2015]. Similarly, digital production facilities have been running certain processes – predominantly long-running batch rendering – in the cloud for years, typically with a significant time and bandwidth cost due to inefficient data transfer [Vanns et al. 2016].

What PipelineX introduces is the idea of backing every transaction and bit of data in the pipeline with a service architecture that can run anywhere in the world. With this simple *combination* of a transactional pipeline and a set of microservices, we can now remotely execute more processes than just rendering, including processes that were previously tricky for us to offload, such as data conversion, data transfer, and simulation. This distinguishes PipelineX from other cloud-based pipelines that are capable of leveraging the cloud but do not natively operate in the cloud.

Additionally, PipelineX aims to do this while still serving up data and tools to the artist in a way that is user-friendly, customizable, and can be utilized on local hosts when necessary. Since the pipeline is agnostic to applications, production management constructs (e.g. sequence/shot), and file formats, we can scale and reshape the pipeline to fit any size and type of production currently on our docket – from a few shots for a television spot to thousands of shots for a ninety-minute feature. We can add new applications, file types, workflows, or even departments whenever we need to, without changing a single service. For us, having such degrees of flexibility is an advantage over cloud-based pipelines that operate with a more limiting set of constraints.

6 BENEFITS

Beyond the ability to remotely process a wealth of new pipeline activities, this microservice-based approach to pipeline has also yielded new levels of information gathering and dependency tracking that we were simply unable to achieve with a more traditional approach. With pipeline logic squarely tucked away in a set of services implemented in an industry-standard way, we get instant access to all the typical benefits of SOA: readily available third-party monitoring solutions, data mining and analytics, fault tolerance, and the ability to scale by spinning up new instances of services to meet production demand. Additionally, we have made the pipeline easier to evolve by mitigating risk of change since development iterations are isolated to subset of affected services.

The event-processing aspect of a microservice architecture lets us tap into the event stream of the pipeline layer to easily notify users when interesting things happen (e.g. a new version of an incoming product is available). We can let users perform the same pipeline activity (e.g. publishing data) from anywhere they'd like – their desks, a dailies room, or eventually mobile devices – simply by sending a snippet of JSON data to a Workflow Service.

Not only does the transactional model work well with our microservice architecture, but it also helps more clearly define responsibility at the artist level, too. It establishes workflow contracts that the pipeline can enforce. Furthermore, it lessens the need for tribal knowledge since all data is strongly typed and tracked explicitly. That, coupled with more dynamic workspace creation, allows us to easily reassign work or transfer data quickly with minimal artist interruption.

7 OUTSTANDING CHALLENGES

A microservice architecture comes with its own set of drawbacks that we are still addressing. Despite the codebase separation inherent in a microservice design, it is fairly easy for a production developer to unwittingly expose the artist to unnecessary levels of detail in the lower-level services, thus ruining our clean separation of technology layers. Additionally, instrumentation and logging in a complex network of interdependent services is challenging; this unfortunately prevents us from immediately improving the daily routine for debugging production problems. Nonetheless, with new debugging tools that leverage the extra metrics we now produce, we believe we can eventually create a better debugging environment in the future. Most challenging of all, however, the abstraction of data has been a tricky proposition for users to accept. It requires careful management of the user experience to ensure artists can still access all types of information about data that they are currently utilizing by navigating a traditional filesystem.

8 CONCLUSION

Years of duct-taping have made our pipeline clunky and more complicated than it really needs to be. This cleaner, more modern architecture is enabling a pipeline that is stable enough to withstand the pressure of modern production schedules, flexible enough to continue pushing the envelope creatively, and scalable enough to meet the ambitions of diversified animation studio.

REFERENCES

Cristian S. Calude, Alasdair Coull, and J. P. Lewis. 2014. Can we solve the pipeline problem?. In *Proceedings of the Fourth Symposium on Digital Production* (DigiPro '14). ACM, New York, NY, USA, 25-27. DOI: http://dx.doi.org/10.1145/2633374.2633380

Chris Johnson, Josef Tobiska, Josh Tomlinson, Nico Van den Bosch, and Wil Whaley. 2014. A framework for global visual effects production pipelines. In *ACM SIGGRAPH 2014 Talks* (SIGGRAPH '14). ACM, New York, NY, USA, Article 57, 1 pages. DOI: https://doi.org/10.1145/2614106.2614159

Bill Polson. 2015. A conceptual framework for pipeline. In *Proceedings of the 2015 Symposium on Digital Production* (DigiPro '15), Stephen Spencer (Ed.). ACM, New York, NY, USA, 51-52. DOI: http://dx.doi.org/10.1145/2791261.2791272

Jim Vanns and Aaron Carey. 2016. A fully cloud-based global visual effects studio. In *ACM SIGGRAPH 2016 Talks* (SIGGRAPH '16). ACM, New York, NY, USA, Article 72, 2 pages. DOI: https://doi.org/10.1145/2897839.2927432