Cloud-Native Data Architecture

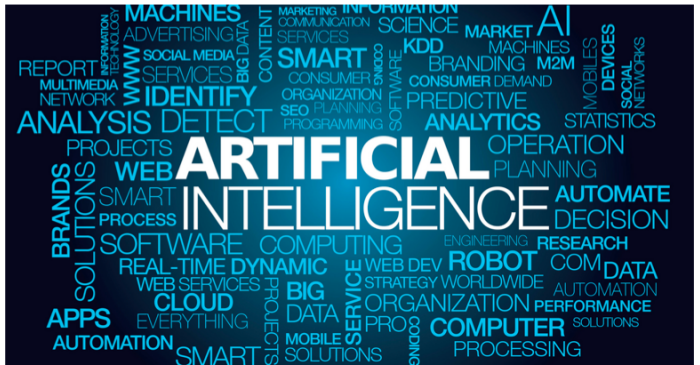
Bart Kersteter – bkersteter@gmail.com

About Me

- I live in Minneapolis, Minnesota
- 20+ years' experience in database administration and architecture
- 6+ years' experience in 'big data'
- Away from work I enjoy:
  - Cooking with my wife
  - Walking my dogs
  - Reading History
  - Playing Tabletop Games
  - Watching sports

The Changing Data Landscape

The world of data processing and data infrastructure is undergoing massive change right now, having a massive impact on what data is processed, how it gets processed and, increasingly, where.



Migrating Workloads to the Cloud

Many companies are starting to migrate their applications to cloud providers like AWS or Azure. There are two approaches to handling this migration:

- Lift & Shift – Run your same systems as-is, just using VMs in the cloud versus VMs or bare metal in your data center.
- Rebuilding your applications to be ‘cloud native’ – i.e. taking advantage of capabilities that cloud-style architecture provides.

What is Cloud Native?

‘Cloud Native’ is about how you design software architectures, not where you deploy them.

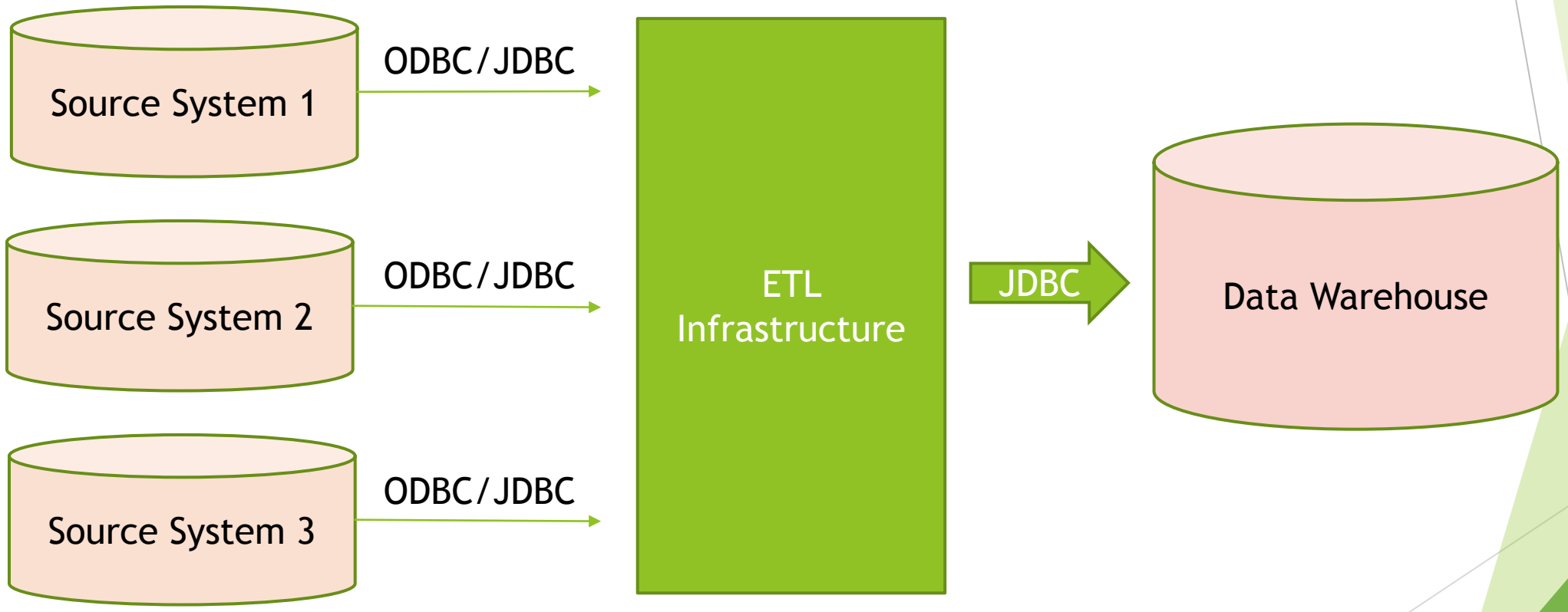
(Public clouds are a great place to run them, though!)

Cloud Native Design Principles: “Assume Fragile”

- Traditional architectures are usually designed assuming that they run entirely in a single data center with known workloads.
- They also use direct connections (a.k.a. “tight coupling”) between systems that often have trouble gracefully reconnecting in case of failures.

- Cloud Native design principles assume a distributed processing environment where failures are occasionally expected and are accounted for in the architecture.

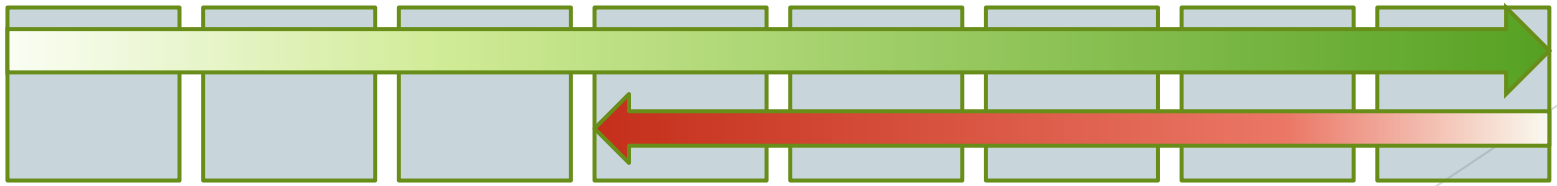
Legacy Data Architecture – "Tight Coupling"



Cloud Native Design Principles: Elasticity

- “Bursty” workloads are common.
- Building the infrastructure to handle peak load on a single day of the year means expensive systems sitting idle for most of the year.
- Designing data systems for the cloud means taking advantage of the horizontal scalability that modern frameworks like MPP databases, Hadoop, Spark and cloud storage offer.

- Cloud Native data architectures are designed to scale up AND down in an automated fashion. You only pay for the resources you need during the time you need them.



Cloud Native Design Principles: “Pets” vs “Cattle”

- Remember when people gave proper names to their servers?
- Traditional infrastructure requires significant resources just to handle security, patching and upgrades, all of which include advance planning, potential risk and downtime.

- Cloud Native pushes a utility computing model, where applications are designed to run on standard infrastructure vs custom patches, OS Libraries, etc.
- Many administrative tasks are handled by the cloud provider
- Pay-by-the hour billing encourages dynamic allocation & deallocation of servers on a daily basis (if not more frequently).

Cloud Native Design Principles: Stateless vs Stateful

Every application needs systems to maintain their state.

Common examples include RDBMS, NoSQL, Message Queues, File Systems/Cloud Storage

Stateless services are common in SOA or Microservice architectures.

- Each instance is autonomous and does not care about any other instance of that service.
- Easily scalable and perfect for container-based architectures.
- Stateless services use an external service registration system like zookeeper, consul or etcd to get needed configuration information.

Cloud Native Design Principles: Streaming AND Batch

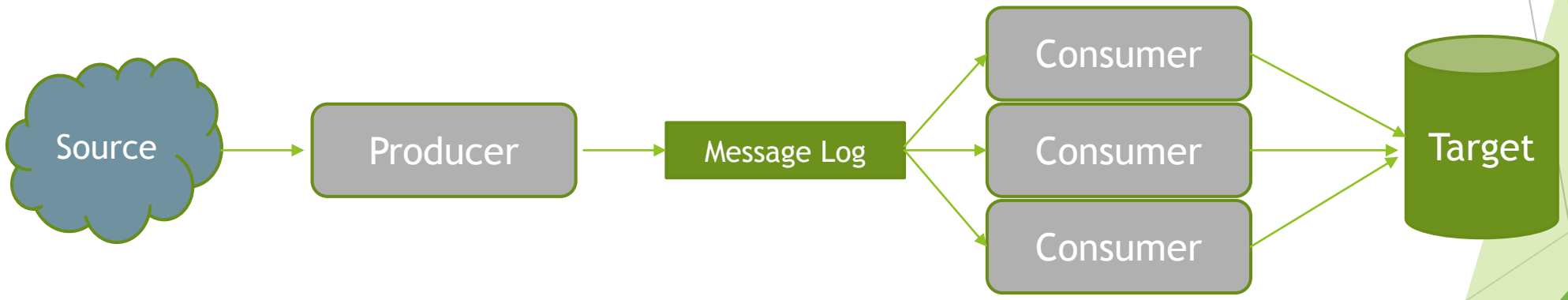
Traditional data integration/ETL systems were aimed primarily at scheduled batch jobs working on regular intervals. The rise of analytics, Internet of Things and social media has seen the rise 24x7x365 streaming ingestion of data. A new paradigm is needed to address that.

Cloud Native designs rely on tools like Apache Kafka or Apache Pulsar to act as the messaging queue or 'unified log' that can handle 'always-on' streaming workloads. These tools also can act as a buffer to handle back-pressure in cases of failures.

Backpressure

“Backpressure” - situations in streaming data architectures where message producers create events faster than message consumers can process them.

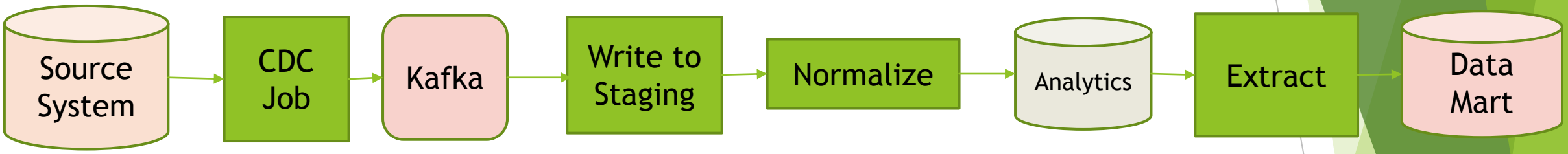
In this scenario, the producer and consumer services are stateless.



If the producer starts generating events faster than the consumer can process them, this leads to queuing in the message log. Add more consumers to alleviate the pressure.

Data Pipelines

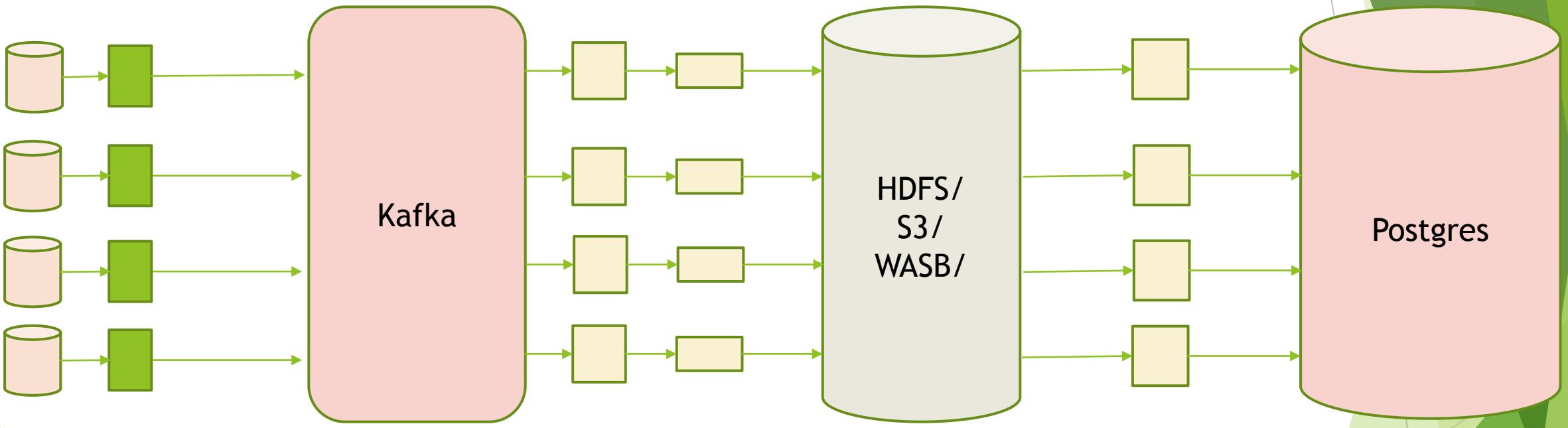
Data pipelines are commonplace in cloud native streaming architectures.



From a process perspective, each data integration workflow is isolated and executed individually. The only dependencies are between the steps in a pipeline for a given job, and the stateful systems that act as sources or targets.

- Stateless services can be scaled & descaled to handle changing workloads.
- Each individual stateless service can be scaled independently
- A message queue like Kafka can scale to contain millions of messages and handle backpressure in the middle of the system.
- In the event of an issue with a stateless service, the service or even virtual servers can be terminated and re-instantiated without having to take the entire streaming workload offline.

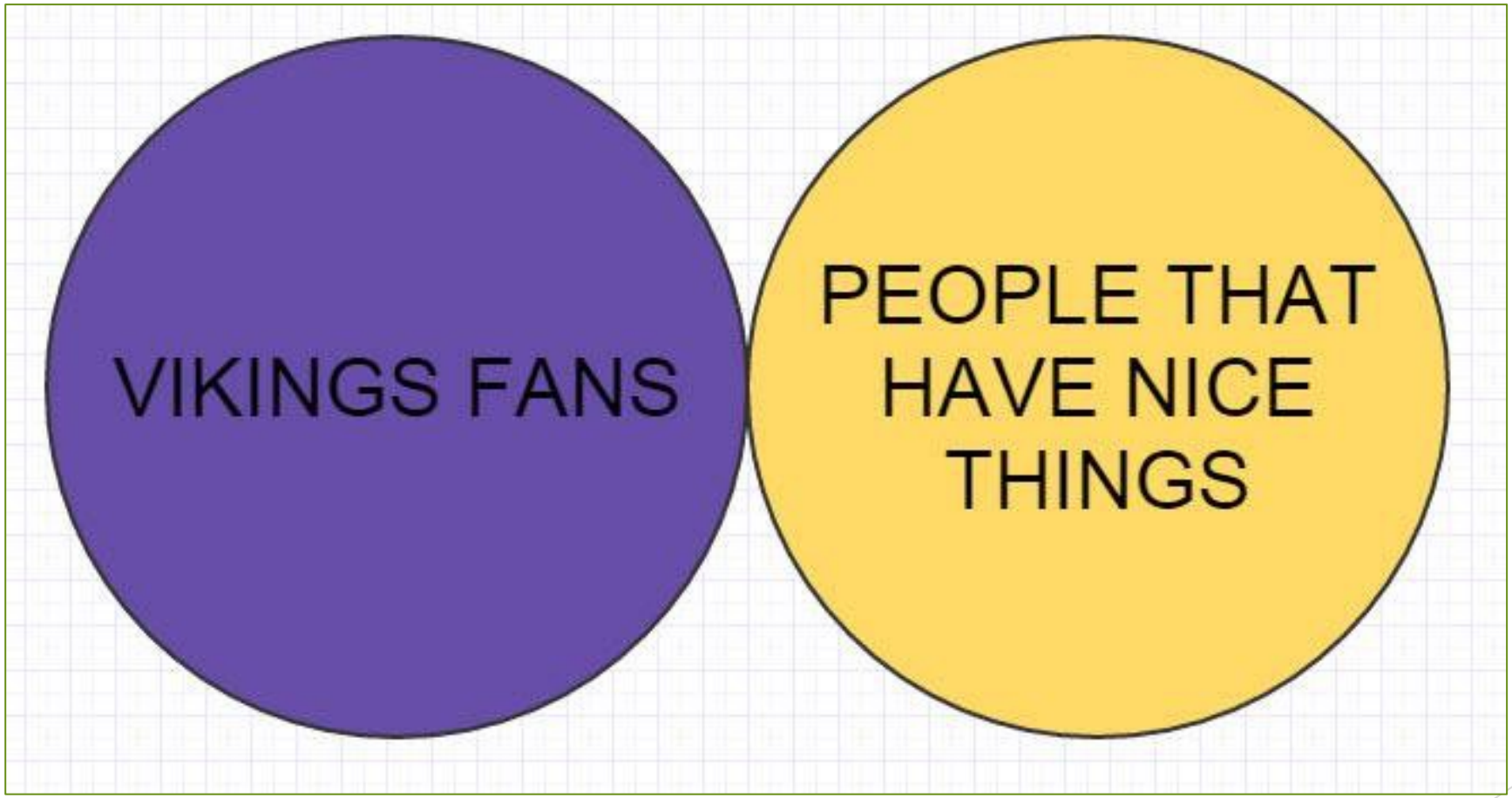
Data Pipelines



At scale, we have individual jobs that use common infrastructure, but are effectively isolated. Adding more pipelines becomes a pure data modeling & infrastructure capacity planning exercise.

Demonstration

Demonstration



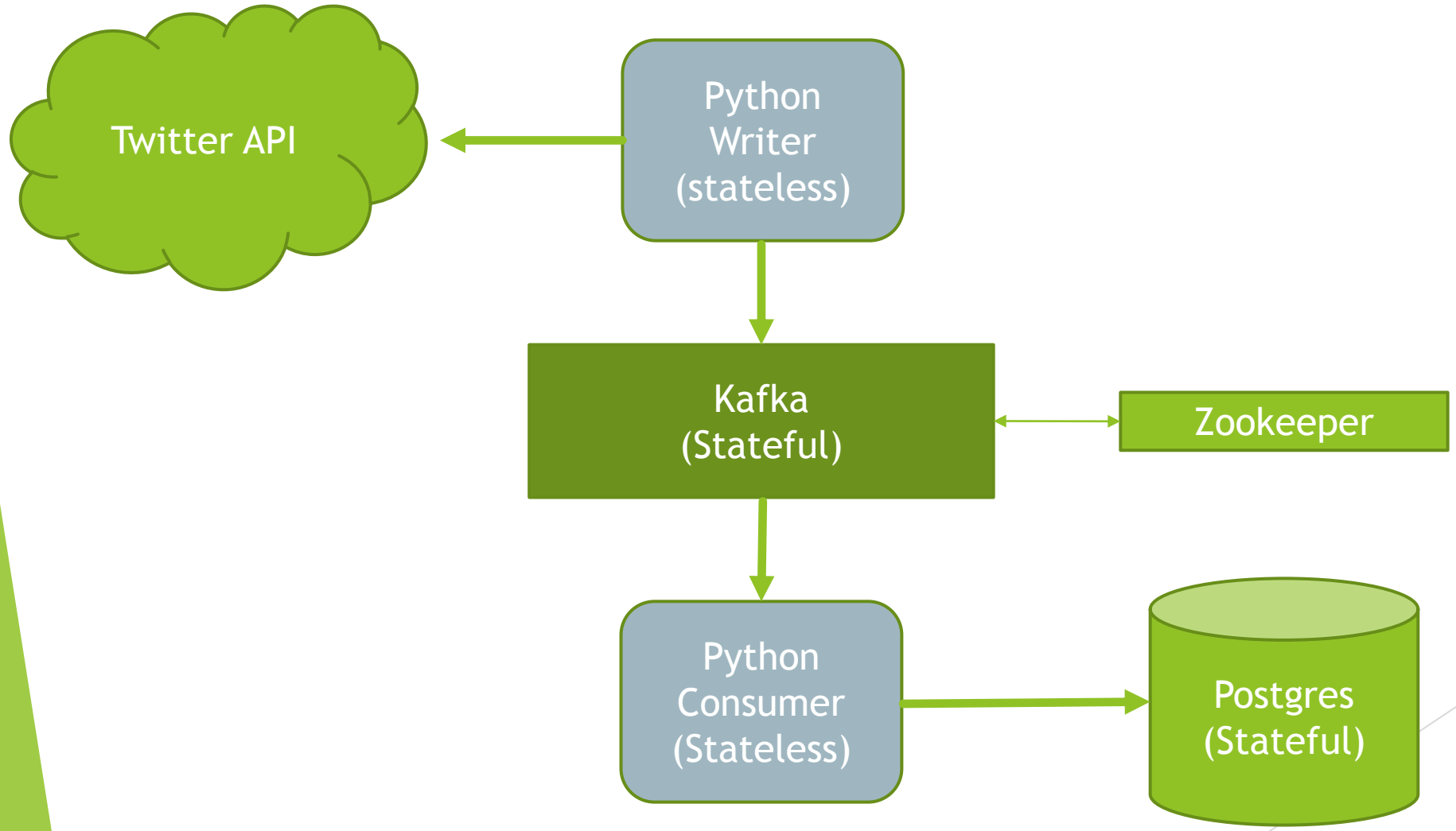
Demonstration

A short demonstration illustrating cloud native design principles in a small system that can run on your desktop or laptop.

Features:

- Python “services” that connect to the Twitter API, search for tweets and then writes to & retrieves messages from a message queue.
- Postgres database acting as the target where selected tweets & associated metadata are stored for analysis.
- Apache Kafka acting as the Message Log that maintains state and handles ”back pressure” in case of Python or Postgres failures.

Demonstration Architecture



Thank You!

Please find this presentation & supporting code at:

https://github.com/bkersteter/cloud-native-demo