Storm Crawler

A real-time distributed web crawling and monitoring framework

Jake Dodd, co-founder

http://ontopic.io

jake@ontopic.io

ApacheCon North America 2015

Agenda

- Overview
- Continuous vs. Batch
- Storm-Crawler Components
- Integration
- Use Cases
- Demonstration
- Q&A

Storm-Crawler

- Software Development Kit (SDK) for building web crawlers on Apache Storm
- https://github.com/DigitalPebble/storm-crawler
- Apache License v2
- Project Director: Julien Nioche (DigitalPebble Ltd)
  - + 3 committers

Facts

- Powered by the Apache Storm framework
- Real-time, distributed, continuous crawling
- Discovery to indexing with low latency
- Java API
- Available as a Maven dependency

The Old Way

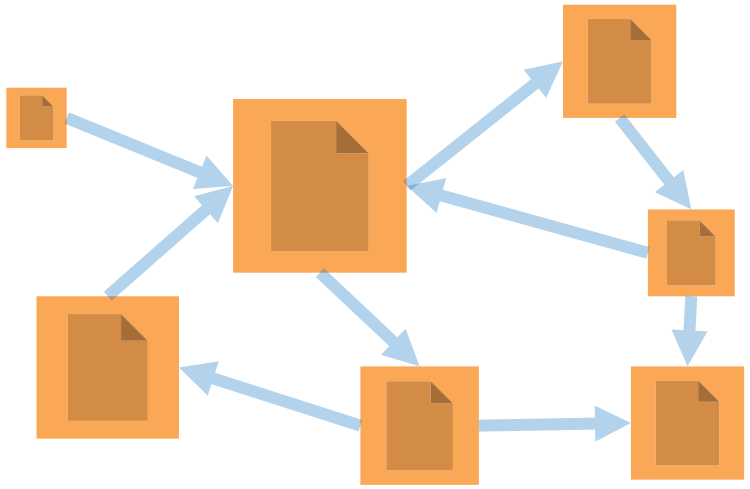
- Batch-oriented crawling
  - Generate a batch of URLs
  - batch fetch → batch parse → batch index → rinse & repeat
- Benefits
  - Well-suited when data locality is paramount
- Challenges
  - Inefficient use of resources—parsing when you could be fetching, hard to allocate and scale resources for individual tasks
  - High latency—at least several minutes, often hours, sometimes *days* between discovery and indexing

Continuous Crawl

- Treat crawling as a *streaming* problem
  - Feed the machine with a stream of URLs, receive a stream of results ASAP
  - URL → fetch → parse → (other stuff) → index
- Benefits
  - Low latency—discovery to indexing in mere moments
  - Efficient use of resources—*always be fetching*
  - Able to allocate resources to tasks on-the-fly (e.g. scale fetchers while holding parsers constant)
  - Easily support **stateful** features (sessions and more)
- Challenges
  - URL queuing and scheduling

The Static Web

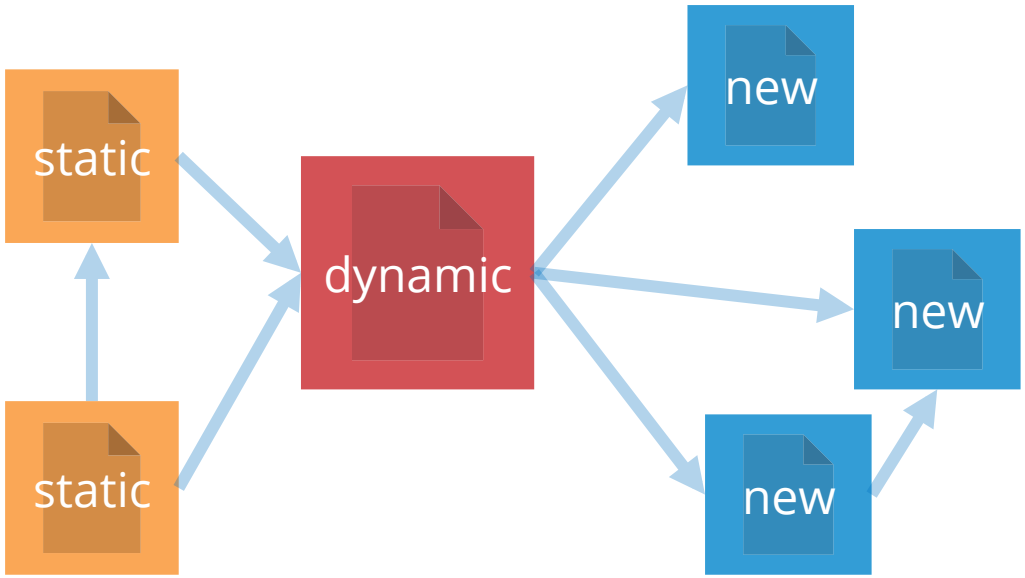
- The Old Model: the web as a collection of linked static documents
  - Still a useful model…just ask Google, Yahoo, Bing, and friends



- But the web has evolved—dynamism is the rule, not the exception

The Web Stream

- Dynamic resources produce a *stream* of links to new documents
  - Applies to web pages, feeds, *and* social media



Can we do both?

- From a crawler's perspective, there's not much difference between *new* and existing (but *newly-discovered*) pages
- Creating a web index from scratch can be modeled as a streaming problem
  - Seed URLs → stream of discovered outlinks → rinse & repeat
- Discovering and indexing new content *is* a streaming problem
- Batch vs. continuous: both methods work, but continuous offers faster data availability
  - Often important for new content

Conclusions

- A modern web crawler should:
  - Use resources efficiently
  - Leverage the elasticity of modern cloud infrastructures
  - Be responsive—fetch and index new documents with low latency
  - Elegantly handle streams of new content
- The dynamic web requires a dynamic crawler

Storm-Crawler: What is it?

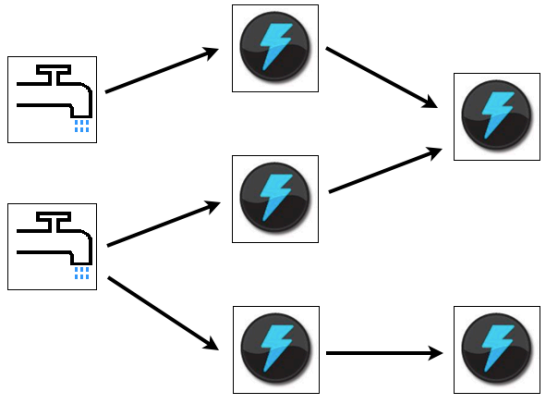
- A Software Development Kit (SDK) for building and configuring continuous web crawlers
- Storm components (spouts & bolts) that handle primary web crawling operations
  - Fetching, parsing, and indexing
- Some of the code has been borrowed (with much gratitude) from Apache Nutch
  - High level of maturity
- Organized into two sub-projects
  - Core (sc-core): components and utilities needed by all crawler apps
  - External (sc-external): components that depend on external technologies (Elasticsearch and more)

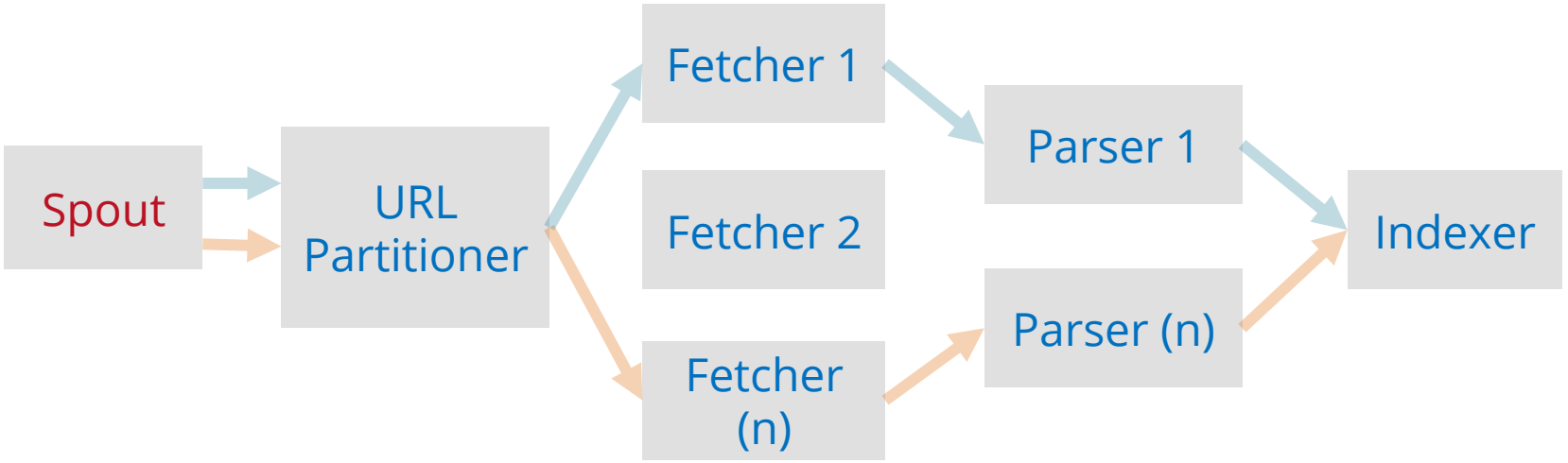
What is it *not*?

- Storm-Crawler is not a full-featured, ready-to-use web crawler application
  - We're in the process of building that separately—will use the Storm-Crawler SDK
- No explicit link & content management (such as linkdb and crawldb with Nutch)
  - But quickly adding components to support recursive crawls
- No PageRank

Basic Topology

Storm topologies consist of spouts and bolts





Spouts

- File spout
  - In sc-core
  - Reads URLs from a file
- Elasticsearch spout
  - In sc-external
  - Reads URLs from an Elasticsearch index
  - Functioning, but we're working on improvements
- Other options (Redis, Kafka, etc.)
  - Will discuss later in presentation

Bolts

- The SDK includes several bolts that handle:
  - URL partitioning
  - Fetching
  - Parsing
  - Filtering
  - Indexing
- We'll briefly discuss each of these

Bolts: URL Partitioner

- Partitions incoming URLs by host, domain, or IP address
  - Strategy is configurable in the topology configuration file
- Creates a partition field in the tuple
  - Storm's grouping feature can then be used to distribute tuples according to requirements
  - localOrShuffle() to randomly distribute URLs to fetchers
  - or fieldsGrouping() to ensure all URLs with the same {host, domain, IP} go to the same fetcher

Bolts: Fetchers

- Two fetcher bolts provided in sc-core
- Both respect robots.txt
- FetcherBolt
  - Multithreaded (configurable number of threads)
  - Use with fieldsGrouping() on the partition key and a configurable crawl delay to ensure your crawler is polite
- SimpleFetcherBolt
  - No internal queues
  - Concurrency configured using parallelism hint and # of tasks
  - Politeness must be handled *outside* of the topology
  - Easier to reason about; requires additional work to enforce politeness

Bolts: Parsers

- Parser Bolt
  - Utilizes Apache Tika for parsing
  - Collects, filters, normalizes, and emits outlinks
  - Collects page metadata (HTTP headers, etc)
  - Parses the page's content to a text representation
- Sitemap Parser Bolt
  - Uses the Crawler-Commons sitemap parser
  - Collects, filters, normalizes, and emits outlinks
  - Requires *a priori* knowledge that a page is a sitemap

Bolts: Indexing

- Printer Bolt (in sc-core)
  - Prints output to stdout—useful for debugging
- Elasticsearch Indexer Bolt (in sc-external)
  - Indexes parsed page content and metadata into Elasticsearch
- Elasticsearch Status Bolt (in sc-external)
  - URLs and their status (discovered, fetched, error) are emitted to a special *status* stream in the storm topology
  - This bolt indexes the URL, metadata, and its status into a 'status' Elasticsearch index

Other components

- URL Filters & Normalizers
  - Configurable with a JSON file
  - Regex filter & normalizer borrowed from Nutch
  - HostURLFilter enables you to ignore outlinks from outside domains or hosts
- Parse Filters
  - Useful for scraping and extracting info from pages
  - XPath-based parse filter, more to come
- Filters & Normalizers are easily pluggable

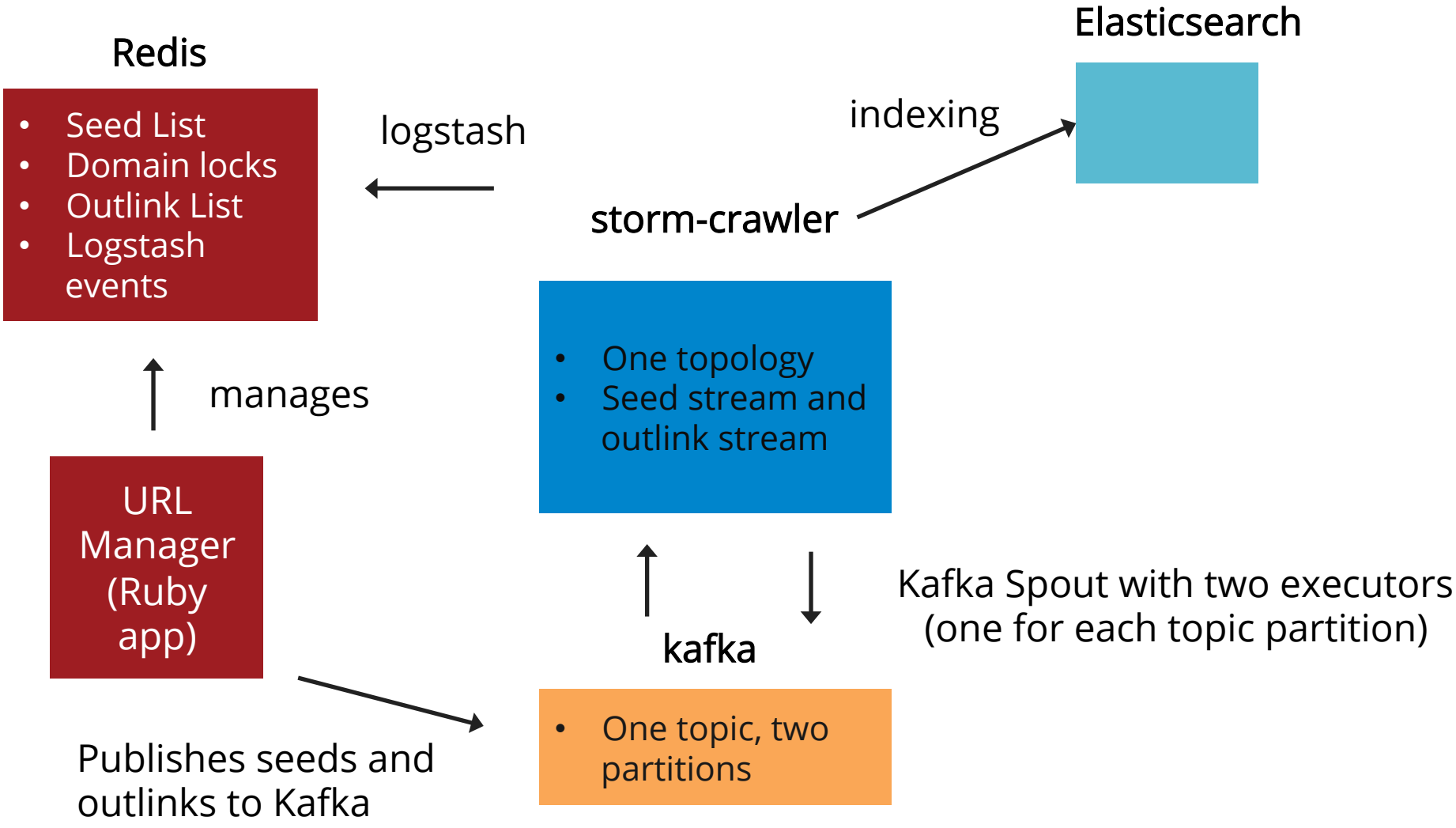
Integrating Storm-Crawler

- Because Storm-Crawler is an SDK, it needs to be integrated with other technologies to build a full-featured web crawler
- At the very least, a database
  - For URLs, metadata, and maybe content
  - Some search engines can double as your core data store (beware…research 'Jepsen tests' for caveats)
- Probably a search engine
  - Solr, Elasticsearch, etc.
  - sc-external provides basic integration with Elasticsearch
- Maybe some distributed system technologies for crawl control
  - Redis, Kafka, ZooKeeper, etc.

Storm-Crawler at Ontopic

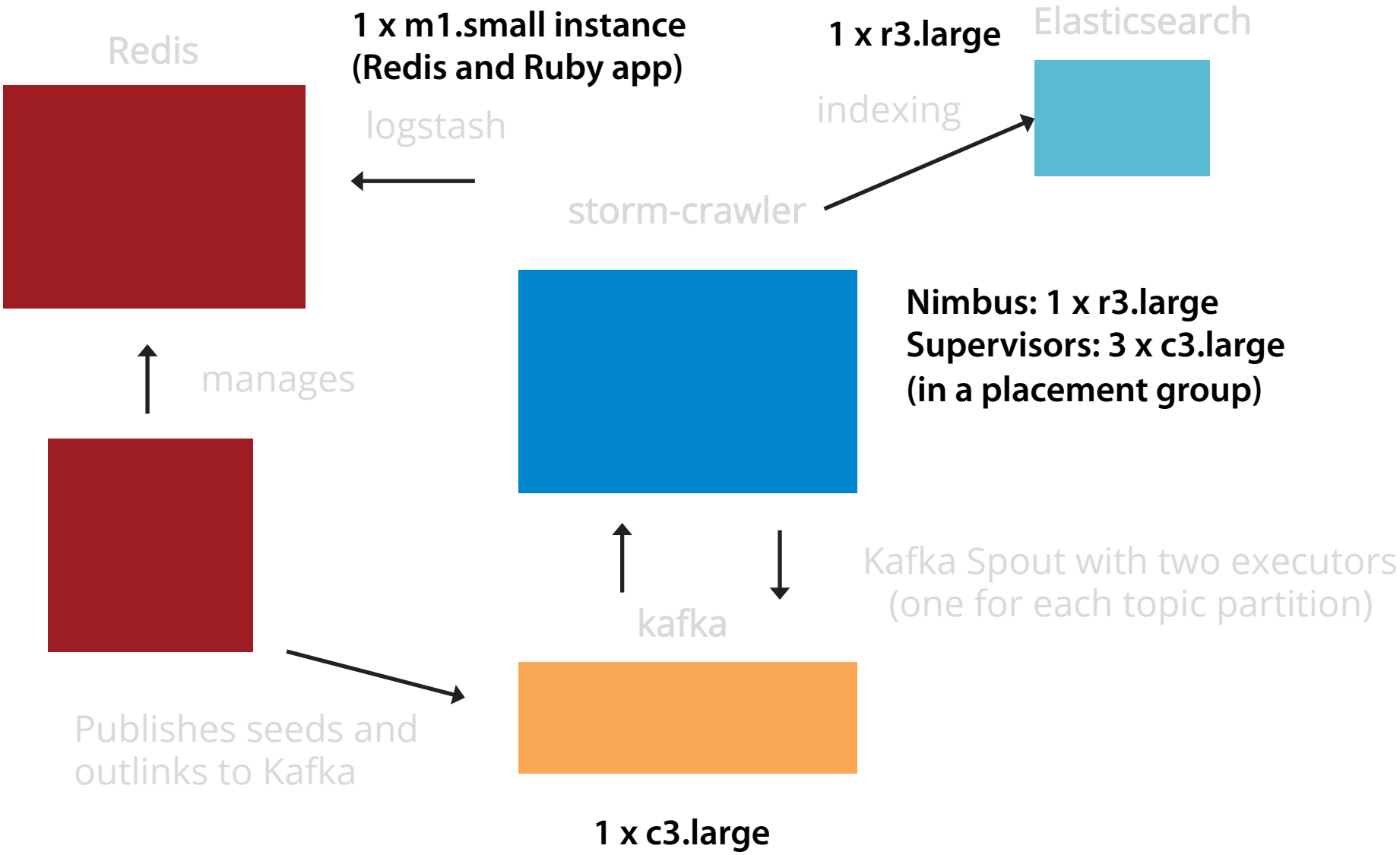
- The storm-crawler SDK is our workhorse for web monitoring
- Integrated with Apache Kafka, Redis, and several other technologies
- Running on an EC2 cluster managed by Hortonworks HDP 2.2

Architecture



R&D Cluster (AWS)

integration



Integration Examples

- Formal crawl metadata specification & serialization with Avro
- Kafka publishing bolt
  - Component to publish crawl data to Kafka (complex URL status handling, for example, could be performed by another topology)
- Externally-stored transient crawl data
  - Components for storing shared crawl data (such as a robots.txt cache) in a key-value store (Redis, Memcached, etc.)

Use Cases & Users

- Processing streams of URLs
  - http://www.weborama.com
- Continuous URL monitoring
  - http://www.shopstyle.com
  - http://www.ontopic.io
- One-off non-recursive crawling
  - http://www.stolencamerafinder.com
- Recursive crawling
  - http://www.shopstyle.com
- More in development & stealth mode

Demonstration

(live demo of Ontopic's topology)

Q&A

Any questions?

Resources

- Project page
  - https://github.com/DigitalPebble/storm-crawler
- Project documentation
  - https://github.com/DigitalPebble/storm-crawler/wiki
- Previous presentations
  - http://www.slideshare.net/digitalpebble/j-nioche-bristoljavameetup20150310
  - http://www.slideshare.net/digitalpebble/storm-crawler-ontopic20141113?related=1
- Other resources
  - http://infolab.stanford.edu/~olston/publications/crawling_survey.pdf

Thank you!