No-Code Queries Can Accelerate AI and Data Analytics

By Dr. Jans Aasman, CEO

The low-code, no-code methodology is becoming highly sought-after throughout the modern IT ecosystem—and with good reason. Options that minimize manually writing code capitalize on the self-service, automation idiom that's imperative in a world in which working remotely and doing more with less keeps organizations in business.

Most codeless or low-code approaches avoid the need for writing language-specific code and replace it with a visual approach in which users simply manipulate on-screen objects via a drag-and-drop, point-and-click interface to automate code generation. The intuitive ease of this approach – which is responsible for new standards of efficiency and democratization of no-code development – has now extended to no-code query writing.

No-code querying provides two unassailable advantages to the enterprise. First, it considerably expedites what is otherwise a time-consuming ordeal, thereby accelerating data analytics and AI-driven applications and second, it can help organizations overcome the talent shortage of developers and knowledge engineers. Moreover, it does so by furnishing all the above benefits that make codeless and low-code options mandatory for success.

Read the full article at DZone.

Data-Centric Architecture Forum – DCAF 2021

Data and the subsequent knowledge derived from information are the most valuable strategic asset an organization possesses. Despite the abundance of sophisticated technology developments, most organizations don't have disciplines or a plan to enable data-centric principles.

DCAF 2021 will help provide clarity.

Our overarching theme for this conference is to **make it REAL**. Real in the sense that others are becoming data-centric, it is achievable, and you are not alone in your efforts.

Join us in understanding how data as an open, centralized resource outlives any application. Once globally integrated by sharing a common meaning, internal and external data can be readily integrated, unlike the traditional "application-centric" mindset predominantly used in systems development.

The compounding problem is these application systems each have their own completely idiosyncratic data models. The net result is that after a few decades, hundreds or thousands of applications implemented have given origin to a segregated family of disparate data silos. Integration debt rises and unsustainable architectural complexity abounds with every application bought, developed, or rented (SaaS).

Becoming data-centric will improve data characteristics of findability, accessibility, interoperability, and re-usability (FAIR principles), thereby allowing data to be exported into any needed format with virtually free integration.\

Dr. Jans Aasman to present – Franz's approach to Entity Event Data Modeling for Enterprise Knowledge Fabrics

---

Text Analytics Forum 2020 – KMWorld Connect

Join us November 17, 2020 – Text Analytics has the ability to add depth, meaning, and intelligence to any organization's most under-utilized resource – text. Through text analytics, enterprises can unlock a wealth of information that would not otherwise be available. Join us as we explore the power of text analytics to provide relevant, valuable, and actionable data for enterprises of all kinds.

Jans Aasman to present – Analyzing Spoken Conversations for Real-Time Decision Support in Mission-Critical Applications

November 17, 2020 at 2PM Eastern

AllegroGraph Named to 100 Companies That Matter Most in Data

Franz Inc. Acknowledged as a Leader for Knowledge Graph Solutions

Lafayette, Calif., June 23, 2020 – Franz Inc., an early innovator in Artificial Intelligence (AI) and leading supplier of Semantic Graph Database technology for Knowledge Graph Solutions, today announced that it has been named to The 100 Companies That Matter in Data by Database Trends and Applications. The annual list reflects the urgency felt among many organizations to provide a timely flow of targeted information. Among the more prominent initiatives is the use of AI and cognitive computing, as well as related capabilities such as machine learning, natural language processing, and text analytics. This list recognizes companies based on their presence, execution, vision and innovation in delivering products and services to the marketplace.

“We’re excited to announce our eighth annual list, as the industry continues to grow and evolve,” remarked Thomas Hogan, Group Publisher at Database Trends and Applications. “Now, more than ever, businesses are looking for ways transform how they operate and deliver value to customers with greater agility, efficiency and innovation. This list seeks to highlight those companies that have been successful in establishing themselves as unique resources for data professionals and stakeholders.”

“We are honored to receive this acknowledgement for our efforts in delivering Enterprise Knowledge Graph Solutions,” said Dr. Jans Aasman, CEO, Franz Inc. “In the past year, we have seen demand for Enterprise Knowledge Graphs take off

across industries along with recognition from top technology analyst firms that Knowledge Graphs provide the critical foundation for artificial intelligence applications and predictive analytics.

Our recent launch of AllegroGraph 7 with FedShard, a breakthrough that allows infinite data integration to unify all data and siloed knowledge into an Entity-Event Knowledge Graph solution will catalyze Knowledge Graph deployments across the Enterprise."

Gartner recently released a report "How to Build Knowledge Graphs That Enable AI-Driven Enterprise Applications" and have previously stated, "The application of graph processing and graph databases will grow at 100 percent annually through 2022 to continuously accelerate data preparation and enable more complex and adaptive data science." To that end, Gartner named graph analytics as a "Top 10 Data and Analytics Trend" to solve critical business priorities. (*Source: Gartner, Top 10 Data and Analytics Trends, November 5, 2019)*.

"Graph databases and knowledge graphs are now viewed as a must-have by enterprises serious about leveraging AI and predictive analytics within their organization," said Dr. Aasman "We are working with organizations across a broad range of industries to deploy large-scale, high-performance Entity-Event Knowledge Graphs that serve as the foundation for AI-driven applications for personalized medicine, predictive call centers, digital twins for IoT, predictive supply chain management and domain-specific Q&A applications – just to name a few."

Forrester Shortlists AllegroGraph

AllegroGraph was shortlisted in the February 3, 2020 Forrester Now Tech: Graph Data Platforms, Q1 2020 report, which recommends that organizations "Use graph data platforms to accelerate connected-data initiatives." Forrester states, "You

can use graph data platforms to become significantly more productive, deliver accurate customer recommendations, and quickly make connections to related data."

Bloor Research covers AllegroGraph with FedShard

Bloor Research Analyst, Daniel Howard noted "With the 7.0 release of AllegroGraph, arguably the most compelling new capability is its ability to create what Franz refers to as "Entity-Event Knowledge Graphs" (or EEKGs) via its patented FedShard technology." Mr. Howard goes on to state "Franz clearly considers this a major release for AllegroGraph. Certainly, the introduction of an explicit entity-event graph is not something I've seen before. The newly introduced text to speech capabilities also seem highly promising."

AllegroGraph Named to KMWorld's 100 Companies That Matter in Knowledge Management

AllegroGraph was also recently named to KMWorld's 100 Companies That Matter in Knowledge Management. The KMWorld 100 showcases organizations that are advancing their products and capabilities to meet changing requirements in Knowledge Management.

Franz Knowledge Graph Technology and Services

Franz's Knowledge Graph Solution includes both technology and services for building industrial strength Entity-Event Knowledge Graphs based on best-of-class tools, products, knowledge, skills and experience. At the core of the solution is Franz's graph database technology, AllegroGraph with FedShard, which is utilized by dozens of the top F500 companies worldwide and enables businesses to extract sophisticated decision insights and predictive analytics from highly complex, distributed data that cannot be uncovered with conventional databases.

Franz delivers the expertise for designing ontology and

taxonomy-based solutions by utilizing standards-based development processes and tools. Franz also offers data integration services from siloed data using W3C industry standard semantics, which can then be continually integrated with information that comes from other data sources. In addition, the Franz data science team provides expertise in custom algorithms to maximize data analytics and uncover hidden knowledge.

Ubiquitous AI Demands A New Type Of Database Sharding

Forbes published the following article by Dr. Jans Aasman, Franz Inc.’s CEO.

The notion of sharding has become increasingly crucial for selecting and optimizing database architectures. In many cases, sharding is a means of horizontally distributing data; if properly implemented, it results in near-infinite scalability. This option enables database availability for business continuity, allowing organizations to replicate databases among geographic locations. It’s equally useful for load balancing, in which computational necessities (like processing) shift between machines to improve IT resource allocation.

However, these use cases fail to actualize sharding’s full potential to maximize database performance in today’s post-big

data landscape. There's an even more powerful form of sharding, called "hybrid sharding," that drastically improves the speed of query results and duly expands the complexity of the questions that can be asked and answered. Hybrid sharding is the ability to combine data that can be partitioned into shards with data that represents knowledge that is usually un-shardable.

This hybrid sharding works particularly well with the knowledge graph phenomenon leveraged by the world's top data-driven companies. Hybrid sharding also creates the enterprise scalability to query scores of internal and external sources for nuanced, detailed results, with responsiveness commensurate to that of the contemporary AI age.

Read the full article at Forbes.

Document Knowledge Graphs with NLP and ML

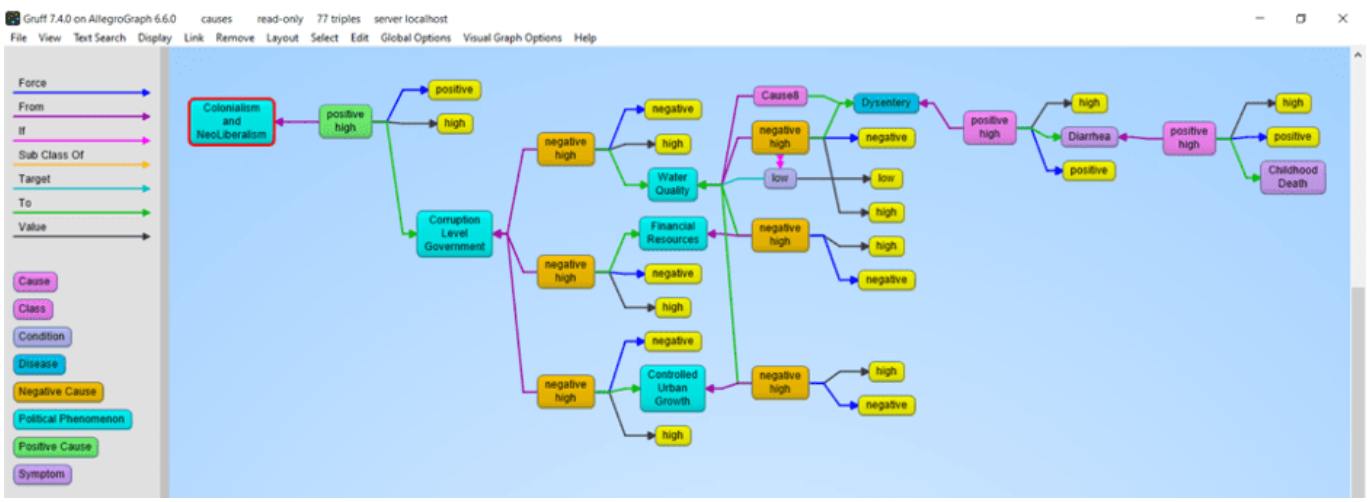
A core competency for Franz Inc is turning text and documents into Knowledge Graphs (KG) using Natural Language Processing (NLP) and Machine Learning (ML) techniques in combination with AllegroGraph. In this document we discuss how the techniques described in [NLP and ML components of AllegroGraph] can be combined with popular software tools to create a robust Document Knowledge Graph pipeline.

We have applied these techniques for several Knowledge Graphs but in this document we will primarily focus on three completely different examples that we summarize below. First is the Chomsky Legacy Project where we have a large set of very dense documents and very different knowledge sources, Second is a knowledge graph for an intelligent call center where we have to deal with high volume dynamic data and real-time decision support and finally, a large government organization where it is very important that people can do a semantic search against documents and policies that steadily change over time and where it is important that you can see the history of documents and policies.

Example [1] Chomsky Knowledge Graph

The Chomsky Legacy Project is a project run by a group of admirers of Noam Chomsky with the primary goal to preserve all his written work, including all his books, papers and interviews but also everything written about him. Ultimately students, researchers, journalists, lobbyists, people from the AI community, and linguists can all use this knowledge graph for their particular goals and questions.

The biggest challenges for this project are finding causal relationships in his work using event and relationship extraction. A simple example we extracted from an author quoting Chomsky is that neoliberalism ultimately causes childhood death.



Example 2: N3 Results and the Intelligent Call Center
This is a completely different use case (See a recent KMWorld Articlehttps://allegrograph.com/knowledge-graphs-enhance-customer-experience-through-speed-and-accuracy/). Whereas the previous use case was very static, this one is highly dynamic. We analyze in real-time the text chats and spoken conversations between call center agents and customers. Our knowledge graph software provides real-time decision support to make the call center agents more efficient. N3 Results helps big tech companies to sell their high tech solutions, mostly cloud-based products and services but also helps their clients sell many other technologies and services.

The main challenge we tackle is to really deeply understand what the customer and agent are talking about. None of this can be solved by only simple entity extraction but requires elaborate rule-based and machine learning techniques. Just to give a few examples. We want to know if the agent talked about their most important talking points: that is, did the agent ask if the customer has a budget, or the authority to make a decision or a timeline about when they need the new technology or whether they actually have expressed their need. But also whether the agent reached the right person, and whether the agent talked about the follow-up. In addition, if the customer talks about competing technology we need to recognize that and provide the agent in real-time with a battle card specific to the competing technology. And in order to be able to do the latter, we also analyzed the complicated marketing materials of the clients of N3.

Example 3: Complex Government Documents
Imagine a regulatory body with tens of thousands of documents. Where nearly every paragraph has reference to other paragraphs in the same document or other documents and the documents change over time. The goal here is to provide the end-users in the government with the right document given their current task at hand. The second goal is to keep track of all the

changes in the documents (and the relationship between documents) over time.

The Document to Knowledge Graph Pipeline

Process Name	Input	Output
1. Custom Taxonomy Creation	Corpus Analytics, Taxonomy tool	A SKOS taxonomy containing concepts, concept hierarchy, prefLabels, altLabels.
2. Document Preparation	Documents (pdf, word, ppt, xlsx), Apache Tika, Spacy for XML cleanup	An XML version of each document
3. Extract Document Meta Data	Document + Apache Tika	JSON dictionary of the Document MetaData
4. XML-to-Triples	XML+JSON dictionary, XMLToTriples.py	Graph-based document tree with chapters, sections, and paragraphs as triples. Also includes meta data as triples
5. Entity-Extraction	Paragraphs + taxonomies + AllegroGraph Entity extract or external extractors	Concepts, persons, places, currencies. Connected to paragraphs
6. LOD Enrichment	Paragraphs + IBM Natural Language Understanding.	Concept categories and links to DBpedia and GeoNames, etc.
7. Complex Relationship and Event extraction.	Paragraphs + Taxonomy + Rules in Spacy or AllegroGraph	Complex events and relationships, References to other document sections.
8. NLP and ML	Chapters and paragraphs + all the tools described [here], but also using Spacy, Gensim, BERT, SciKit Learn.	Similarities, sentiment, query answering, smart search, text classification, word embeddings, abstracts
9. Versioning and Document tracking	Old + New document, compare.py	Old document in historic repository, new document in current, changed graph.
10. Statistical Relationships	Concepts + OddRatio.py or OddsRatio.cl	Statistical relationships between concepts.

Let us first give a quick summary in words of how we turn documents into a Knowledge Graph.

[1] Taxonomy Creation

Taxonomy of all the concepts important to the business using open source or commercial taxonomy builders. An available industry taxonomy is a good starting point for additional customizations.

[2] Document Preparation

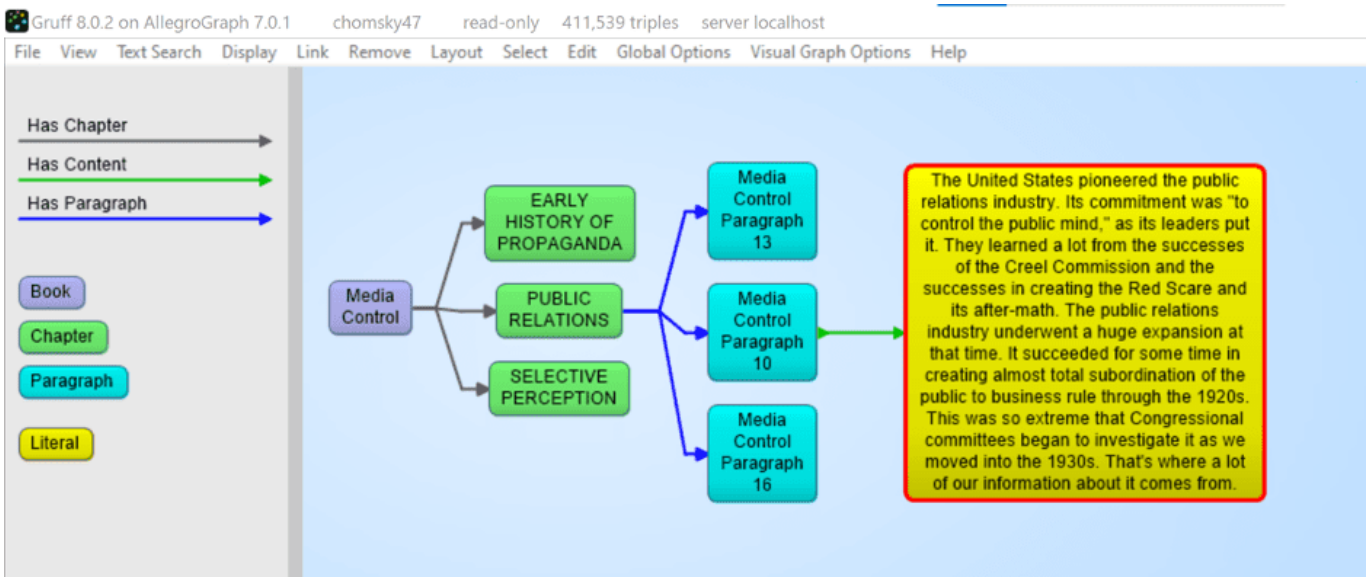
We then take a document and turn it into an intermediate XML using Apache Tika. Apache Tika supports more than 1000 document types and although Apache Tika is a fantastic tool, the output is still usually not clean enough to create a graph from, so we use Spacy rules to clean up the XML to make it as uniform as possible.

[3] Extract Document MetaData

Most documents also contain document metadata (author, date, version, title, etc) and Apache Tika will also deliver the metadata for a document as a JSON object.

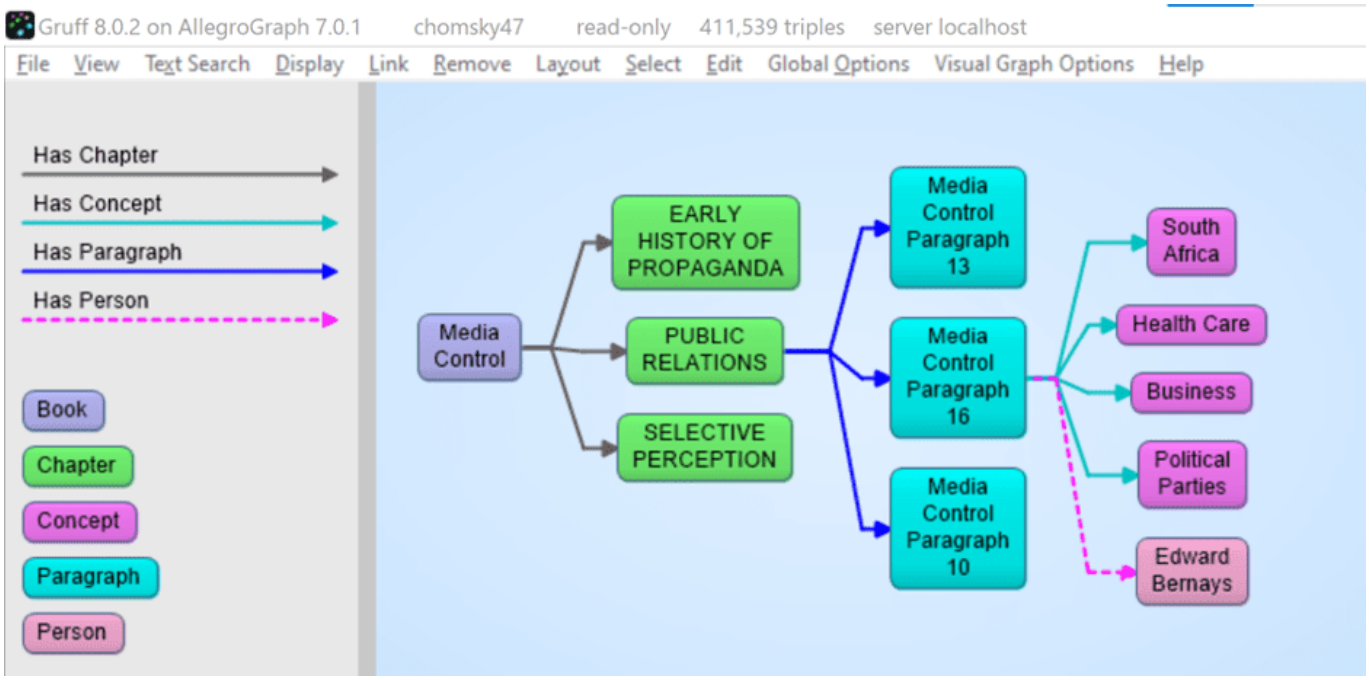
[4] XML to Triples

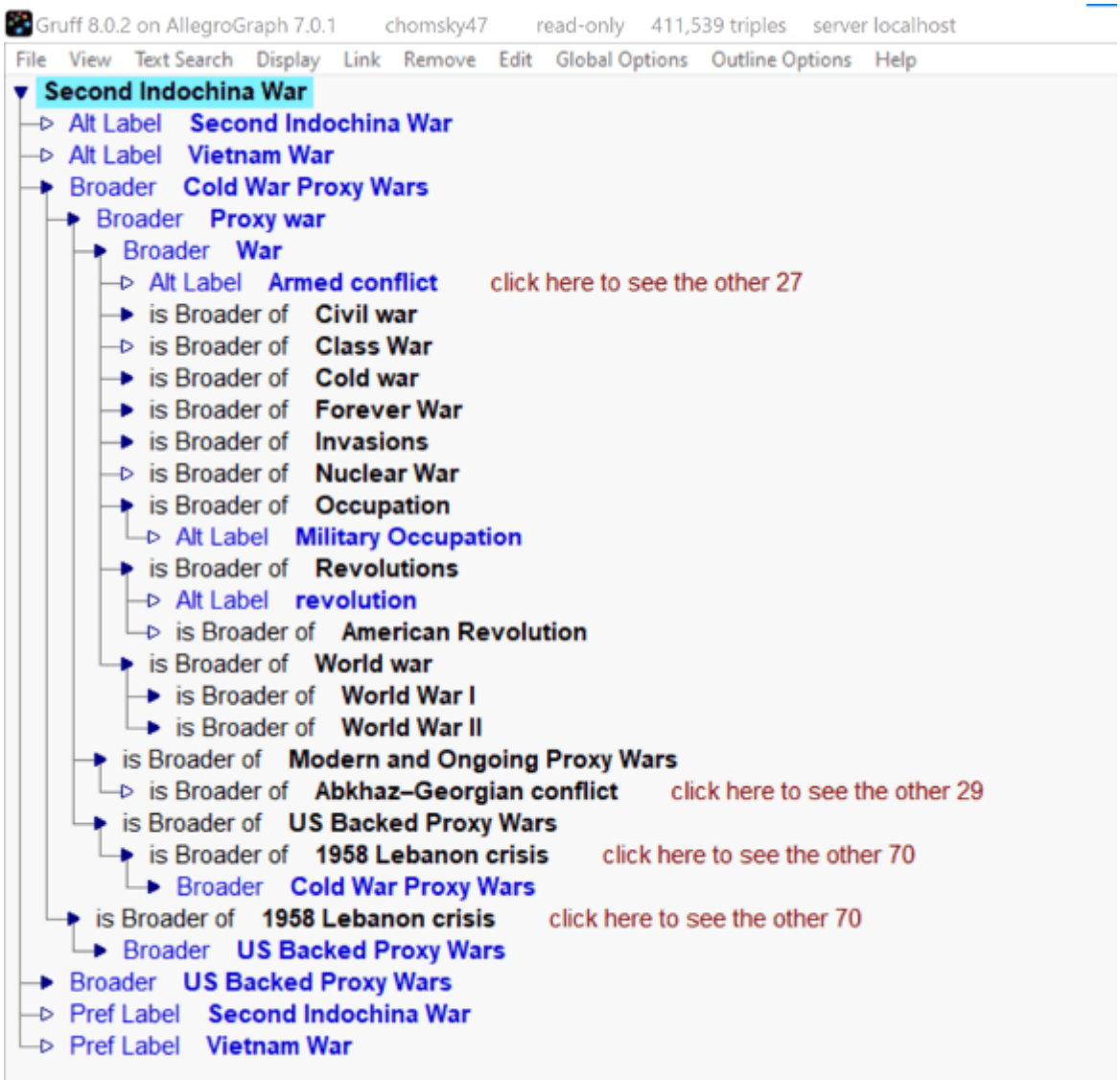
Our tools ingest the XML and metadata and transform that into a graph-based document tree. The document is the root and from that, it branches out into chapters, optionally sections, all the way down to paragraphs. The ultimate text content is in the paragraphs. In the following example we took the XML version of Noam Chomsky's book Media Control and turned that into a tree. The following shows a tiny part of that tree. We start with the Media Control node, then we show three (of the 11) chapters, for one chapter we show three (of the 6) paragraphs, and then we show the actual text in that paragraph. We sometimes can go even deeper to the level of sentences and tokens but for most projects that is overkill.



[5] Entity Extractor

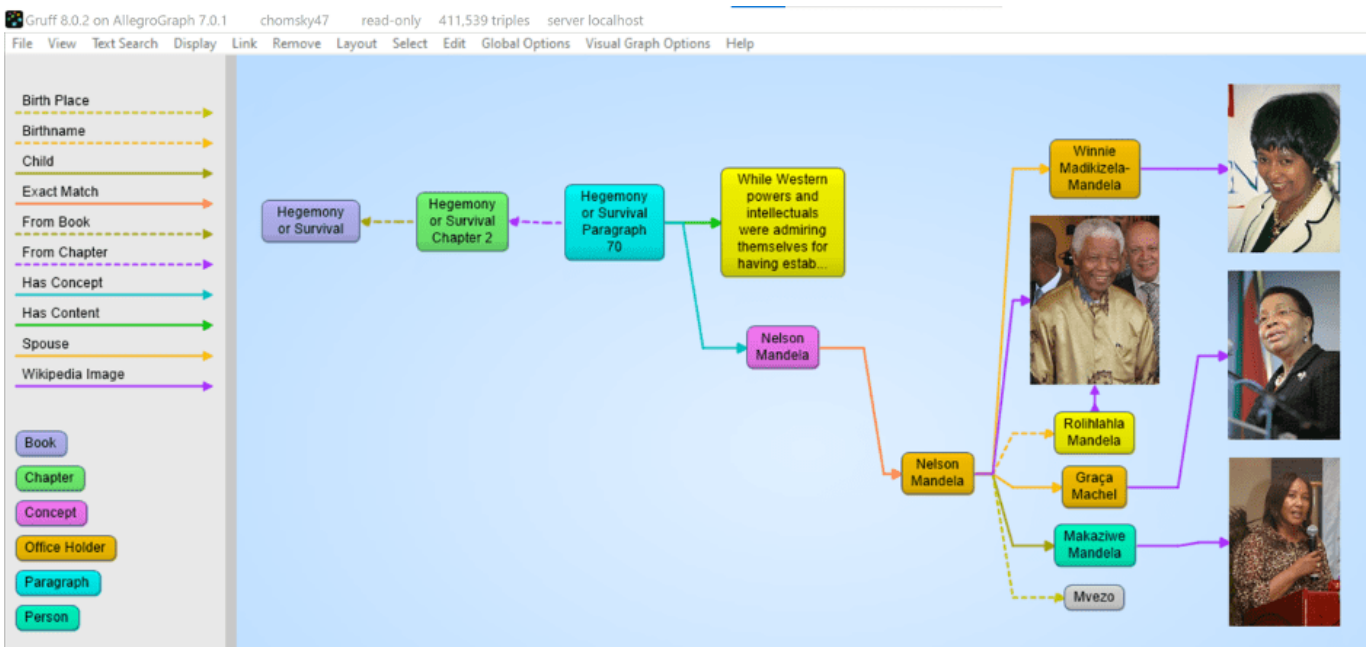
AllegroGraph's entity extractor takes as input the text of each paragraph in the document tree and one or more of the taxonomies and returns recognized SKOS concepts based on prefLabels and altLabels. AllegroGraph's entity extractor is state of the art and especially powerful when it comes to complex terms like product names. We find that in our call center a technical product name can sometimes have up to six synonyms or very specific jargon. For example the Cisco product Catalyst 9000 will also be abbreviated as the cat 9k. Instead of developing altLabels for every possible permutation that human beings *will* use, we have specialized heuristics to optimize the yield from the entity extractor. The following picture shows 4 (of the 14) concepts discovered in paragraph 16. Plus one person that was extracted by IBM's NLU.





[6] Linked Data Enrichment

In many use cases, AllegroGraph can link extracted entities to concepts in the linked data cloud. The most prominent being DBpedia, wikidata, the census database, GeoNames, but also many Linked Open Data repositories. One tool that is very useful for this is IBM's Natural Language Understanding program but there are others available. In the following image we see that the Nelson Mandela entity (Red) is linked to the dbpedia entity for Nelson Mandela and that then links to the DBpedia itself. We extracted some of his spouses and a child with their pictures.



[7] Complex Relationship and Event Extraction

Entity extraction is a first good step to 'see' what is in your documents but it is just the first step. For example: how do you find in a text whether company C1 merged with company C2. There are many different ways to express the fact that a company fired a CEO. For example: Uber got rid of Kalanick, Uber and Kalanick parted ways, the board of Uber kicked out the CEO, etc. We need to write explicit symbolic rules for this or we need a lot of training data to feed a machine learning algorithm.

[8] NLP and Machine Learning

There are many many AI algorithms that can be applied in Document Knowledge Graphs. We provide best practices for topics like:

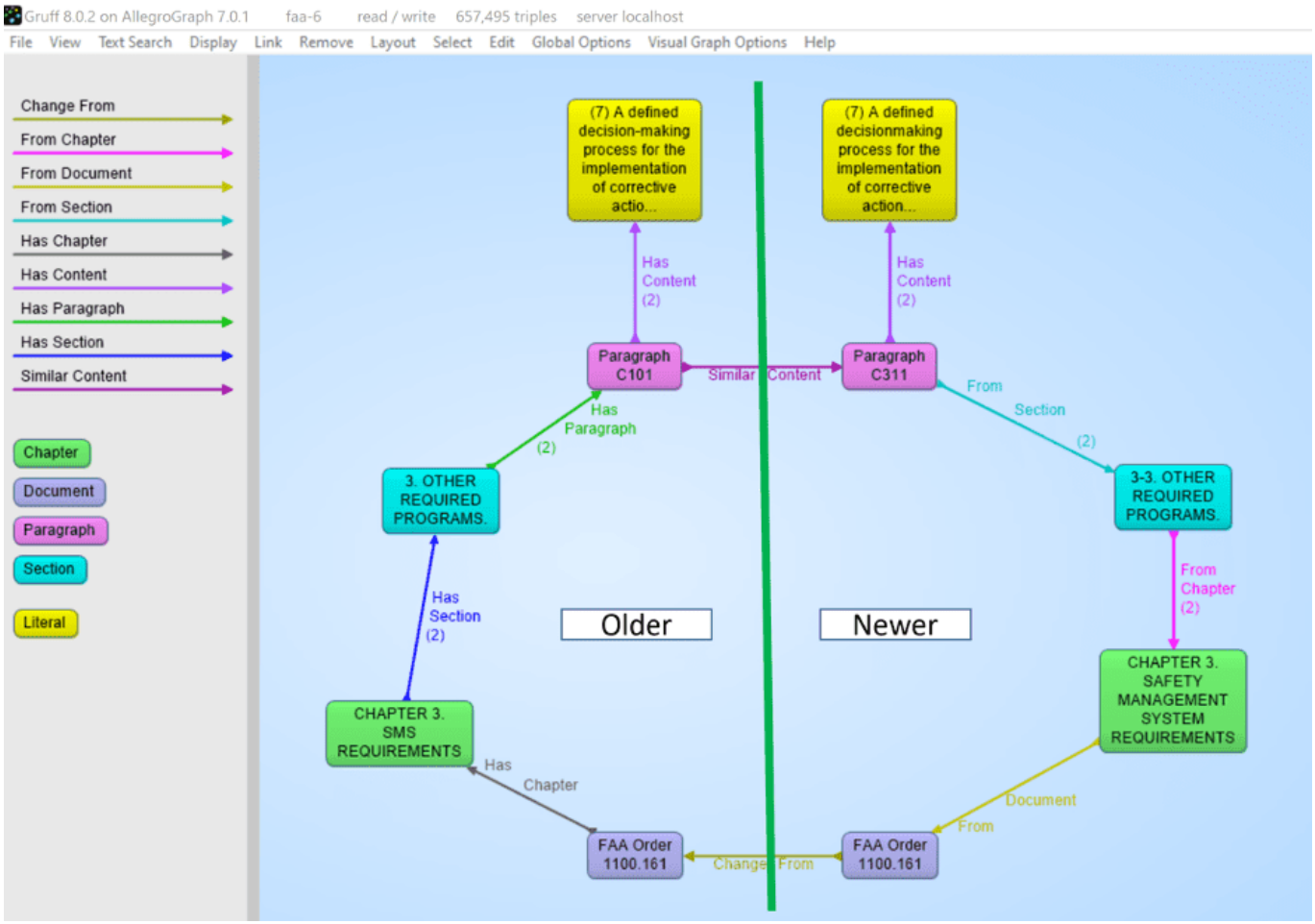
[a] Sentiment Analysis, using good/bad word lists or training data.
[b] Paragraph or Chapter similarity using statistical techniques like Gensim similarity or symbolic techniques where we just the overlap of recognized entities as a function of the size of a text.
[c] Query answering using word2vec or more advanced techniques like BERT
[d] Semantic search using the hierarchy in SKOS taxonomies.
[e] Summarization techniques for Abstractive or Extractive abstracts using Gensim or Spacy.

[9] Versioning and Document tracking

Several of our customers with Document Knowledge Graphs have noted the one constant in all of these KGs is that documents change over time. As part of our solution, we have created best practices where we deal with these changes. A crucial first step is to put each document in its own graph (i.e. the fourth element of every triple in the document tree is the document id itself). When we get a new version of a document the document ID changes but the new document will point back to the old version. We then compute which paragraphs stayed the same within a certain margin (there are always changes in whitespace) and we materialize what paragraphs disappeared in the new version and what new paragraphs appeared compared to the previous version. Part of the best practice is to put the old version of a document in a historical database that at all times can be federated with the ‘current’ set of documents.

Note that in the following picture we see the progression of a document. On the right hand side we have a newer version of a document 1100.161 with a chapter -> section -> paragraph -> contents where the content is almost the same as the one in

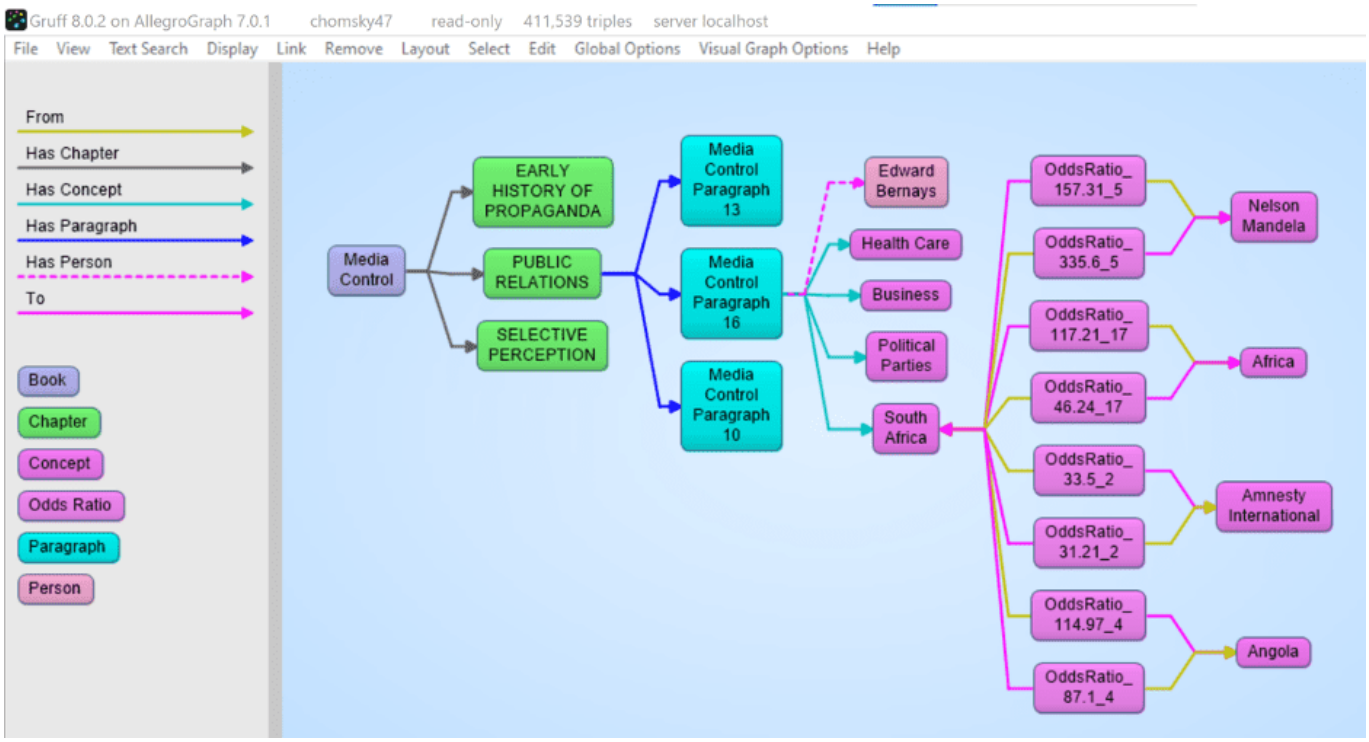
the older version. But note that the newer one spells ‘decision making’ as one word whereas the older version said ‘decision-making’. Note that also the chapter titles and the section titles are almost the same but not entirely. Also, note that the new version has a back-pointer (changed-from) to the older version.



[10] Statistical Relationships

One important analytic one can do on documents is to look at the co-occurrence of terms. Although, given that certain words might occur more frequently in text, we have to correct the co-occurrence between words for the frequency of the two terms in a co-occurrence to get a better idea of the ‘surprisingness’ of a co-occurrence. The platform offers several techniques in Python and Lisp to compute these co-occurrences. Note that in the following picture we computed the odds ratios between recognized entities and so we see in

the following gruff picture that if Noam Chomsky talks about South Africa then the chances are very high he will also talk about Nelson Mandela.



The Knowledge Graph Cookbook

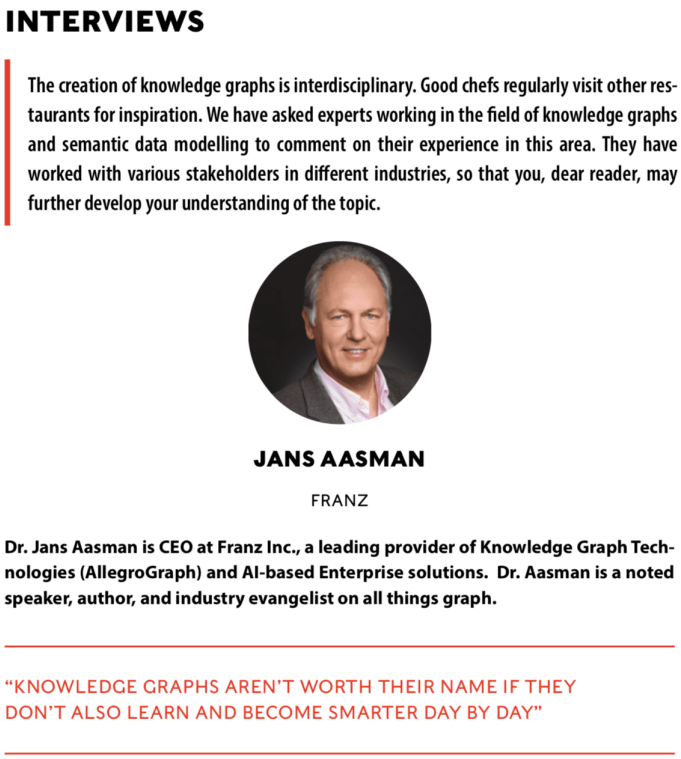
Recipes for Knowledge Graphs that Work:

- Learn why and how to build knowledge graphs that help enterprises use data to innovate, create value and increase revenue. This practical manual is full of recipes and knowledge on the subject.
- Learn more about the variety of applications based on knowledge graphs.
- Learn how to build working knowledge graphs and which technologies to use.
- See how knowledge graphs can benefit different parts of your organization.

- Get ready for the next generation of enterprise data management tools.

Dr. Jans Aasman, CEO, Franz Inc. is interviewed in the Expert Opinion Section.

> **“KNOWLEDGE GRAPHS AREN’T WORTH THEIR NAME IF THEY DON’T ALSO LEARN AND BECOME SMARTER DAY BY DAY”** – Dr. Aasman



INTERVIEWS

The creation of knowledge graphs is interdisciplinary. Good chefs regularly visit other restaurants for inspiration. We have asked experts working in the field of knowledge graphs and semantic data modelling to comment on their experience in this area. They have worked with various stakeholders in different industries, so that you, dear reader, may further develop your understanding of the topic.

JANS AASMAN

FRANZ

Dr. Jans Aasman is CEO at Franz Inc., a leading provider of Knowledge Graph Technologies (AllegroGraph) and AI-based Enterprise solutions. Dr. Aasman is a noted speaker, author, and industry evangelist on all things graph.

“KNOWLEDGE GRAPHS AREN’T WORTH THEIR NAME IF THEY DON’T ALSO LEARN AND BECOME SMARTER DAY BY DAY”

Click here to get the book as free PDF or Kindle version.

California utilities should have used digital twin technology instead of power shutoffs

Northern California's proactive power outages were not necessary last fall. Digital Twin technology can predict utility line failures and turn off power in milliseconds to avoid the potential of sparks igniting the surrounding area.

Digital twin technologies are gaining traction across industries and use cases. Initially devised as a means of monitoring assets and production settings in manufacturing, this technology has quietly seeped into other verticals like hospitality, construction, and building management and soon, electricity delivery.

The premier problem digital twins will solve is predicting power grid failure, which would alleviate the social, economic, and political issues that resulted from efforts to reduce the incidence and degree of catastrophes, property loss, and deaths stemming from downstream effects of power grid failure—such as recurring wildfires.

Digital twins can allay these concerns because they're based on real-time signals from a comprehensive set of factors that could be indicative of power grid woes related to environmental, meteorological, or technology concerns. Moreover, they can deliver accurate predictions for each of these factors well in advance of failure—in some cases as much as 28 days.

Read the full article at PowerGrid International.

Franz Inc. to Present at The Global Graph Summit and Data Day Texas

Dr. Jans Aasman, CEO, Franz Inc., will be presenting, “Creating Explainable AI with Rules” at the Global Graph Summit, a part of Data Day Texas. The abstract for Dr. Aasman’s presentation:

> *“There’s a fascinating dichotomy in artificial intelligence between statistics and rules, machine learning and expert systems. Newcomers to artificial intelligence (AI) regard machine learning as innately superior to brittle rules-based systems, while the history of this field reveals both rules and probabilistic learning are integral components of AI. This fact is perhaps nowhere truer than in establishing* explainable AI, *which is central to the long-term business value of AI front-office use cases.”*

> *“The fundamental necessity for explainable AI spans regulatory compliance, fairness, transparency, ethics and lack of bias – although this is not a complete list. For example, the effectiveness of counteracting financial crimes and increasing revenues from advanced machine learning predictions in financial services could be greatly enhanced by deploying more accurate deep learning models. But all of this would be arduous to explain to regulators. Translating those results into explainable rules is the basis for more widespread AI deployments producing a more meaningful impact on society.”*

The Global Graph Summit is an independently organized vendor-neutral conference, bringing leaders from every corner of the graph and linked-data community for sessions, workshops, and its well-known before and after parties. Originally launched in January 2011 as one of the first NoSQL / Big Data conferences, Data Day Texas each year highlights the latest tools, techniques, and projects in the data space, bringing speakers and attendees from around the world to enjoy the hospitality that is uniquely Austin. Since its inception, Data Day Texas has continually been the largest independent data-centric event held within 1000 miles of Texas.

Multi-Master Replication Clusters in Kubernetes and Docker Swarm

For more examples visit – https://github.com/franzinc/agraph-examples

Introduction

In this document we primarily discuss running a Multi-Master Replication cluster (MMR) inside Kubernetes. We will also show a Docker Swarm implementation.

This directory and subdirectories contain code you can use to run an MMR cluster. The second half of this document is entitled *Setting up and running MMR under Kubernetes* and that is where you'll see the steps needed to run the MMR cluster in Kubernetes.

MMR replication clusters are different from distributed AllegroGraph clusters in these important ways:

1. Each member of the cluster needs to be able to make a TCP connection to each other member of the cluster. The connection is to a port computed at run time. The range of port numbers to which a connection is made can be constrained by the agraph.cfg file but typically this will be a large range to ensure that at least one port in that range is not in used.
2. All members of the cluster hold the complete database (although for brief periods of time they can be out of sync and catching up with one another).

MMR replication clusters don't quite fit the Kubernetes model in these ways

1. When the cluster is running normally each instance knows the DNS name or IP address of each other instance. In Kubernetes you don't want to depend on the IP address of another cluster's pod as those pods can go away and a replacement started at a different IP address. We'll describe below our solution to this.
2. Services are a way to hide the actual location of a pod however they are designed to handle a set of known ports.. In our case we need to connect from one pod to a known-at-runtime port of another pod and this isn't what services are designed for.

3. A key feature of Kubernetes is the ability to scale up and down the number of processes in order to handle the load appropriately. Processes are usually single purpose and stateless. An MMR process is a full database server with a complete copy of the repository. Scaling up is not a quick and simple operation – the database must be copied from another node. Thus scaling up is a more deliberate process rather than something automatically done when the load on the system changes during the day.

The Design

1. We have a headless service for our controlling instance StatefulSet and that causes there to be a DNS entry for the name *controlling* that points to the current IP address of the node in which the controlling instance runs. Thus we don't need to hardwire the IP address of the controlling instance (as we do in our AWS load balancer implementation).
2. The controlling instance uses two PersistentVolumes to store: 1. The repo we're replicating and 2. The token that other nodes can use to connect to this node. Should the controlling instance AllegroGraph server die (or the pod in which it runs dies) then when the pod is started again it will have access to the data on those two persistent volumes.
3. We call the other instances in the cluster Copy instances. These are full read-write instances of the repository but we don't back up their data in a persistent volume. This is because we want to scale up and down the number of Copy instances. When we scale down we don't want to save the old data since when we scale down we remove that instance from the cluster thus the repo in the cluster can never join the cluster again. We denote the Copy instances by their IP addresses. The Copy instances can find the address of the controlling instance via DNS. The controlling

instance will pass the cluster configuration to the Copy instance and that configuration information will have the IP addresses of the other Copy instances. This is how the Copy instances find each other.

4. We have a load balancer that allows one to access a random Copy instance from an external IP address. This load balancer doesn’t support sessions so it’s only useful for doing queries and quick inserts that don’t need a session.
5. We have a load balancer that allows access to the Controlling instance via HTTP. While this load balancer also doesn’t have session support, because there is only one controlling instance it’s not a problem if you start an AllegroGraph session because all sessions will live on the single controlling instance.

We’ve had the most experience with Kubernetes on the Google Cloud Platform. There is no requirement that the load balancer support sessions and the GCP version does not at this time, but that doesn’t mean that session support isn’t present in the load balancer in other cloud platforms. Also there is a large community of Kubernetes developers and one may find a load balancer with session support available from a third party.

Implementation

We build and deploy in three subdirectories. We’ll describe the contents of the directories first and then give step by step instructions on how to use the contents of the directories.

Directory ag/

In this directory we build a Docker image holding an installed AllegroGraph. The Dockerfile is

```
FROM centos:7
```

```
#
# AllegroGraph root is /app/agraph
#

RUN yum -y install net-tools iputils bind-utils wget hostname

ARG agversion=agraph-6.6.0
ARG agdistfile=${agversion}-linuxamd64.64.tar.gz

# This ADD command will automatically extract the contents
# of the tar.gz file
ADD ${agdistfile} .

# needed for agraph 6.7.0 and can't hurt for others
# change to 11 if you only have OpenSSL 1.1 installed
ENV ACL_OPENSSL_VERSION=10

# so prompts are readable in an emacs window
ENV PROMPT_COMMAND=

RUN groupadd agraph && useradd -d /home/agraph -g agraph
agraph
RUN mkdir /app

# declare ARGs as late as possible to allow previous lines to
be cached
# regardless of ARG values

ARG user
ARG password

RUN (cd ${agversion} ;  ./install-agraph /app/agraph -- --non-
interactive \
                --runas-user agraph \
                --super-user $user \
                --super-password $password )

# remove files we don't need
RUN rm -fr /app/agraph/lib/doc /app/agraph/lib/demos

# we will attach persistent storage to this directory
```

```
VOLUME ["/app/agraph/data/rootcatalog"]

# patch to reduce cache time so we'll see when the controlling instance moves.
# ag 6.7.0 has config parameter StaleDNSRetainTime which allows this to be
# done in the configuration.
COPY dnspatch.cl /app/agraph/lib/patches/dnspatch.cl

RUN chown -R agraph.agraph /app/agraph
```

The Dockerfile installs AllegroGraph in /app/agraph and creates an AllegroGraph super user with the name and password passed in as arguments. It creates a user *agraph* so that the AllegroGraph server will run as the user *agraph* rather than as *root*.

We have to worry about the controlling instance process dying and being restarted in another pod with a different IP address. Thus if we've cached the DNS mapping of *controlling* we need to notice as soon as possible that the mapping as changed. The dnspatch.cl file changes a parameter in the AllegroGraph DNS code to reduce the time we trust our DNS cache to be accurate so that we'll quickly notice if the IP address of *controlling* changes.

We also install a number of networking tools. AllegroGraph doesn't need these but if we want to do debugging inside the container they are useful to have installed.

The image created by this Dockerfile is pushed to the Docker Hub using an account you've specified (see the Makefile in this directory for details).

Directory agrepl/

Next we take the image created above and add the specific code to support replication clusters.

The Dockerfile is

```
ARG DockerAccount=specifyaccount

FROM ${DockerAccount}/ag:latest

#
# AllegroGraph root is /app/agraph

RUN mkdir /app/agraph/scripts
COPY . /app/agraph/scripts

# since we only map one port from the outside into our cluster
# we need any sessions created to continue to use that one port.
RUN echo "UseMainPortForSessions true" >> /app/agraph/lib/agraph.cfg

# settings/user will be overwritten with a persistent mount so copy
# the data to another location so it can be restored.
RUN cp -rp /app/agraph/data/settings/user /app/agraph/data/user

ENTRYPOINT ["/app/agraph/scripts/repl.sh"]
```

When building an image using this Dockerfile you must specify

```
--build-arg DockerAccount=MyDockerAccount
```

where MyDockerAccount is a Docker account you're authorized to push images to.

The Dockerfile installs the scripts repl.sh, vars.sh and accounts.sh. These are run when this container starts.

We modify the agraph.cfg with a line that ensures that even if we create a session that we'll continue to access it via port 10035 since the load balancer we'll use to access AllegroGraph only forwards 10035 to AllegroGraph.

Also we know that we'll be installing a persistent volume

at /app/agraph/data/user so we make a copy of that directory in another location since the current contents will be invisible when a volume is mounted on top of it. We need the contents as that is where the credentials for the user we created when AllegroGraph was installed.

Initially the file settings/user/*username* will contain the credentials we specified when we installed AllegroGraph in first Dockerfile. When we create a cluster instance a new token is created and this is used in place of the password for the test account. This token is stored in settings/user/*username* which is why we need this to be an instance-specific and persistent filesystem for the controlling instance.

When this container starts it runs repl.sh which first runs accounts.sh and vars.sh.

accounts.sh is a file created by the top level Makefile to store the account information for the user account we created when we installed AllegroGraph.

vars.sh is

```
# constants need by scripts
port=10035
reponame=myrepl

# compute our ip address, the first one printed by hostname
myip=$(hostname -I | sed -e 's/ .*$//')
```

In vars.sh we specify the information about the repository we'll create and our IP address.

The script repl.sh is this:

```
#!/bin/bash
#
## to start ag and then create or join a cluster
##
```

```
cd /app/agraph/scripts

set -x
. ./accounts.sh
. ./vars.sh

agtool=/app/agraph/bin/agtool

echo ip is $myip


# move the copy of user with our login to the newly mounted
volume
# if this is the first time we've run agraph on this volume
if [ ! -e /app/agraph/data/rootcatalog/$reponame ] ; then
                  cp    -rp    /app/agraph/data/user/*
/app/agraph/data/settings/user
fi

# due to volume mounts /app/agraph/data could be owned by root
# so we have to take back ownership
chown -R agraph.agraph /app/agraph/data


## start agraph
/app/agraph/bin/agraph-control                --config
/app/agraph/lib/agraph.cfg start

term_handler() {
    # this signal is delivered when the pod is
    # about to be killed.  We remove ourselves
    # from the cluster.
   echo got term signal
   /bin/bash ./remove-instance.sh
   exit
}

sleepforever() {
    # This unusual way of sleeping allows
    # a TERM signal sent when the pod is to
    # die to then cause the shell to invoke
```

```
    # the term_handler function above.
    date
    while true
    do
        sleep 99999 & wait ${!}
    done
}

if [ -e /app/agraph/data/rootcatalog/$reponame ] ; then
     echo repository $reponame already exists in this
persistent volume
    sleepforever
fi

controllinghost=controlling

controllingspec=$authuser:$authpassword@$controllinghost:$port
/$reponame

if [ x$Controlling == "xyes" ] ;
then
    # It may take a little time for the dns record for
'controlling' to be present
   # and we need that record because the agtool program below
will use it
   until host controlling ; do  echo controlling not in DNS
yet; sleep 5 ; done
   ## create first and controlling cluster instance
   $agtool repl create-cluster $controllingspec controlling

else
    # wait for the controlling ag server to be running
                                until       curl       -s
http://$authuser:$authpassword@$controllinghost:$port/version
; do echo wait for controlling ; sleep 5; done

    # wait for server in this container to be running
                                until       curl       -s
http://$authuser:$authpassword@$myip:$port/version ; do echo
wait for local server ; sleep 5; done
```

```
   # wait for cluster repo on the controlling instance to be
present
   until $agtool repl status $controllingspec > /dev/null ; do
echo wait for repo ; sleep 5; done
   myiname=i-$myip
   echo $myiname > instance-name.txt

   # construct the remove-instance.sh shell script to remove
this instance
   # from the cluster when the instance is terminated.
    echo $agtool repl remove $controllingspec $myiname >
remove-instance.sh
   chmod 755 remove-instance.sh
   #

   # note that
   #  % docker kill container
   # will send a SIGKILL signal by default  we can't trap on
SIGKILL.
   # so
   #  % docker kill -s TERM container
   # in order to test this handler
   trap term_handler SIGTERM SIGHUP SIGUSR1
   trap -p
   echo this pid is $$

   # join the cluster
   echo joining the cluster
       $agtool repl grow-cluster $controllingspec
$authuser:$authpassword@$myip:$port/$reponame $myiname
fi
sleepforever
```

This script can be run under three different conditions

1. Run when the Controlling instance is starting for the first time
2. Run when the Controlling instance is restarting having run before and died (perhaps the machine on which it was running crashed or the AllegroGraph process had some

error)
3. Run when a Copy instance is starting for the first time. Copy instances are not restarted when they die. Instead a new instance is created to take the place of the dead instance. Therefore we don't need to handle the case of a Copy instance restarting.

In cases 1 and 2 the environment variable *Controlling* will have the value "yes".

In case 2 there will be a directory at /app/agraph/data/rootcatalog/$reponame.

In all cases we start an AllegroGraph server.

In case 1 we create a new cluster. In case 2 we just sleep and let the AllegroGraph server recover the replication repository and reconnect to the other members of the cluster.

In case 3 we wait for the controlling instance's AllegroGraph to be running. Then we wait for our AllegroGraph server to be running. Then we wait for the replication repository we want to copy to be up and running. At that point we can grow the cluster by copying the cluster repository.

We also create a script which will remove this instance from the cluster should this pod be terminated. When the pod is killed (likely due to us scaling down the number of Copy instances) a termination signal will be sent first to the process allowing it to run this remove script before the pod completely disappears.

Directory kube/

This directory contains the yaml files that create kubernetes resources which then create pods and start the containers that create the AllegroGraph replication cluster.

controlling-service.yaml

We begin by defining the services. It may seem logical to define the applications before defining the service to expose the application but it’s the service we create that puts the application’s address in DNS and we want the DNS information to be present as soon as possible after the application starts. In the repl.sh script above we include a test to check when the DNS information is present before allowing the application to proceed.

```
apiVersion: v1
kind: Service
metadata:
 name: controlling
spec:
 clusterIP:  None
 selector:
   app: controlling
 ports:
 - name: http
   port: 10035
   targetPort: 10035
```

This selector defines a service for any container with a label with a key app and a value controlling. There aren’t any such containers yet but there will be. You create this service with

```
% kubectl create -f controlling-service.yaml
```

In fact for all the yaml files shown below you create the object they define by running

```
% kubectl create -f  filename.yaml
```

copy-service.yaml

We do a similar service for all the copy applications.

```
apiVersion: v1
kind: Service
metadata:
 name: copy
spec:
```

```
 clusterIP: None
 selector:
   app: copy
 ports:
 - name: main
   port: 10035
   targetPort: 10035
```

controlling.yaml

This is the most complex resource description for the cluster. We use a StatefulSet so we have a predictable name for the single pod we create. We define two persistent volumes. A StatefulSet is designed to control more than one pod so rather than a VolumeClaim we have a VolumeClaimTemplate so that each Pod can have its own persistent volume… but as it turns out we have only one pod in this set and we never scale up. There must be exactly one controlling instance.

We setup a liveness check so that if the AllegroGraph server dies Kubernetes will restart the pod and thus the AllegroGraph server. Because we've used a persistent volume for the AllegroGraph repositories when the AllegroGraph server restarts it will find that there is an existing MMR replication repository that was in use when the AllegroGraph server was last running. AllegroGraph will restart that replication repository which will cause that replication instance to reconnect to all the copy instances and become part of the cluster again.

We set the environment variable Controlling to yes and this causes this container to start up as a controlling instance (you'll find the check for the Controlling environment variable in the repl.sh script above).

We have a volume mount for /dev/shm, the shared memory filesystem, because the default amount of shared memory allocated to a container by Kubernetes is too small to support AllegroGraph.

```
#
# stateful set of controlling instance
#

apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: controlling
spec:
  serviceName: controlling
  replicas: 1
  template:
    metadata:
      labels:
        app: controlling
    spec:
        containers:
        - name: controlling
          image: dockeraccount/agrepl:latest
          imagePullPolicy: Always
          livenessProbe:
            httpGet:
              path: /hostname
              port: 10035
            initialDelaySeconds: 30
          volumeMounts:
          - name: shm
            mountPath: /dev/shm
          - name: data
            mountPath: /app/agraph/data/rootcatalog
          - name: user
            mountPath: /app/agraph/data/settings/user
          env:
          - name: Controlling
            value: "yes"
        volumes:
         - name: shm
           emptyDir:
             medium: Memory
  volumeClaimTemplates:
         - metadata:
```

```
            name: data
          spec:
            resources:
              requests:
                storage: 20Gi
            accessModes:
            - ReadWriteOnce
        - metadata:
            name: user
          spec:
            resources:
              requests:
                storage: 10Mi
            accessModes:
            - ReadWriteOnce
```

copy.yaml

This StatefulSet is responsible for starting all the other instances. It’s much simpler as it doesn’t use Persistent Volumes

```
#
# stateful set of copies of the controlling instance
#

apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: copy
spec:
  serviceName: copy
  replicas: 2
  template:
    metadata:
      labels:
        app: copy
    spec:
        volumes:
        - name: shm
          emptyDir:
            medium: Memory
```

```
        containers:
        - name: controlling
          image: dockeraccount/agrepl:latest
          imagePullPolicy: Always
          livenessProbe:
            httpGet:
              path: /hostname
              port: 10035
            initialDelaySeconds: 30
          volumeMounts:
          - name: shm
            mountPath: /dev/shm
```

controlling-lb.yaml

We define a load balancer so applications on the internet outside of our cluster can communicate with the controlling instance. The IP address of the load balancer isn’t specified here. The cloud service provider (i.e. Google Cloud Platform or AWS) will determine an address after a minute or so and will make that value visible if you run

```
% kubectl get svc controlling-loadbalancer
```

The file is

```
apiVersion: v1
kind: Service
metadata:
  name: controlling-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 10035
    targetPort: 10035
  selector:
    app: controlling
```

copy-lb.yaml

As noted earlier the load balancer for the copy instances does not support sessions. However you can use the load balancer to

issue queries or simple inserts that don’t require a session.

```
apiVersion: v1
kind: Service
metadata:
  name: copy-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 10035
    targetPort: 10035
  selector:
    app: copy
```

copy-0-lb.yaml

If you wish to access one of the copy instances explicitly so that you can create sessions you can create a load balancer which links to just one instance, in this case the first copy instance which is named “copy-0”.

```
apiVersion: v1
kind: Service
metadata:
  name: copy-0-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 10035
    targetPort: 10035
  selector:
    app: copy
    statefulset.kubernetes.io/pod-name: copy-0
```

Setting up and running MMR under Kubernetes

The code will build and deploy an AllegroGraph MMR cluster in Kubernetes. We’ve tested this in Google Cloud Platform and Amazon Web Service. This code requires Persistent Volumes and load balancers and thus requires a sophisticated platform to run (such as GCP or AWS).

Prerequisites

In order to use the code supplied you'll need two additional things

1. A Docker Hub account (https://hub.docker.com). A free account will work. You'll want to make sure you can push to the hub without needing a password (use the docker login command to set that up).
2. An AllegroGraph distribution in tar.gz format. We've been using agraph-6.6.0-linuxamd64.64.tar.gz in our testing. You can find the current set of server files at https://franz.com/agraph/downloads/server This file should be put in the ag subdirectory. Note that the Dockerfile in that directory has the line ARG agversion=agraph-6.6.0 which specifies the version of agraph to install. This must match the version of the ...tar.gz file you put in that directory.

Steps

Do Prerequisites

Fullfill the prerequisites above

Set parameters

There are 5 parameters

1. Docker account – **Must Specify**
2. AllegroGraph user – **May want to specify**
3. AllegroGraph password – **May want to specify**
4. AllegroGraph repository name – **Unlikely to want to change**
5. AllegroGraph port – **Very unlikely to want to change**

The first three parameters can be set using the Makefile in the top level directory. The last two parameters are found in agrepl/vars.sh if you wish to change them. Note that the port number of 10035 is found in the yaml files in

the kube subdirectory. If you change the port number you'll have edit the yaml files as well.

The first three parameters are set via

```
% make account=DockerHubAccount user=username password=password
```

The account must be specified but the last two can be omitted and default to an AllegroGraph account name of *test* and a password of *xyzzy*.

If you choose to specify a password make it a simple one consisting of letters and numbers. The password will appear in shell commands and URLs and our simple scripts don't escape characters that have a special meaning to the shell or URLs.

Install AllegroGraph

Change to the ag directory and build an image with AllegroGraph installed. Then push it to the Docker Hub

```
% cd ag
% make build
% make push
% cd ..
```

Create cluster-aware AllegroGraph image

Add scripts to create an image that will either create an AllegroGraph MMR cluster or join a cluster when started.

```
% cd agrepl
% make build
% make push
% cd ..
```

Setup a Kubernetes cluster

Now everything is ready to run in a Kubernetes cluster. You may already have a Kubernetes cluster running or you may need to create one. Both Google Cloud Platform and AWS have ways of

creating a cluster using a web UI or a shell command. When you've got your cluster running you can do

```
% kubectl get nodes
```

and you'll see your nodes listed. Once this works you can move into the next step.

Run an AllegroGraph MMR cluster

Starting the MMR cluster involves setting up a number of services and deploying pods. The Makefile will do that for you.

```
% cd kube
% make doall
```

You'll see when it displays the services that there isn't an external IP address allocated for the load balancers It can take a few minutes for an external IP address to be allocated and the load balancers setup so keep running

```
% kubectl  get svc
```

until you see an IP address given, and even then it may not work for a minute or two after that for the connection to be made.

Verify that the MMR cluster is running

You can use AllegroGraph Webview to see if the MMR cluster is running. Once you have an external IP address for the controlling-load-balancer go to this address in a web browser

```
http://external-ip-address:10035
```

Login with the credentials you used when you created the Docker images (the default is user *test* and password *xyzzy*). You'll see a repository myrepl listed. Click on that. Midway down you'll see a link titled

Manage Replication Instances as controller

Click on that link and you'll see a table of three instances which now serve the same repository. This verifies that three pods started up and all linked to each other.

Namespaces

All objects created in Kubernetes have a name that is chosen either by the user or Kubernetes based on a name given by the user. Most names have an associated namespace. The combination of namespace and name must be unique among all objects in a Kubernetes cluster. The reason for having a namespace is that it prevents name clashes between multiple projects running in the same cluster that both choose to use the same name for an object.

The default namespace is named default.

Another big advantage using namespaces is that if you delete a namespace you delete all objects whose name is in that namespace. This is useful because a project in Kubernetes uses a lot of different types of objects and if you want to delete all the objects you've added to a Kubernetes cluster it can take a while to find all the objects by type and then delete them. However if you put all the objects in one namespace then you need only delete the namespace and you're done.

In the Makefile we have this line

Namespace=testns

which is used by this rule

```
reset:
        -kubectl delete namespace ${Namespace}
        kubectl create namespace ${Namespace}
         kubectl config set-context `kubectl config current-
context` --namespace ${Namespace}
```

The reset rule deletes all members of the Namespace named at

the top of the Makefile (here testns) and then recreates the namespace and switches to it as the active namespace. After doing the reset all objects created will be created in the testns namespace.

We include this in the Makefile because you may find it useful.

Docker Swarm

The focus of this document is Kubernetes but we also have a Docker Swarm implementation of an AllegroGraph MMR cluster. Docker Swarm is significantly simpler to setup and manage than Kubernetes but has far fewer bells and whistles. Once you've gotten the ag and agrepl images built and pushed to the Docker Hub you need only link a set of machines running Docker together into a Docker Swarm and then

% cd swarm ; make controlling copy

and the AllegroGraph MMR cluster is running Once it is running you can access the cluster using Webview at

http://localhost:10035/