Databases on AWS: How To Choose The Right Database

Randall Hunt
Software Engineer at AWS
@jrhunt
randhunt@amazon

Markus Ostertag
CEO at Team Internet AG
@Osterjour

Dr. Sebastian Brandt
Senior Key Expert - Knowledge Graph at Siemens CT
A Quick History of Databases
Modern apps have modern requirements

Users: 1 million+
Data volume: TB–PB–EB
Locality: Global
Performance: Milliseconds–microseconds
Request rate: Millions per second
Access: Web, Mobile, IoT, Devices
Scale: Up-down, Out-in
Economics: Pay for what you use
Developer access: No assembly required
## Common data categories and use cases

<table>
<thead>
<tr>
<th>Relational</th>
<th>Key-value</th>
<th>Document</th>
<th>In-memory</th>
<th>Graph</th>
<th>Time-series</th>
<th>Ledger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referential integrity, ACID transactions, schema-on-write</td>
<td>High throughput, low-latency reads and writes, endless scale</td>
<td>Store documents and quickly query on any attribute</td>
<td>Query by key with microsecond latency</td>
<td>Quickly and easily create and navigate relationships between data</td>
<td>Collect, store, and process data sequenced by time</td>
<td>Complete, immutable, and verifiable history of all changes to application data</td>
</tr>
</tbody>
</table>

- **Lift and shift, ERP, CRM, finance**
- **Real-time bidding, shopping cart, social, product catalog, customer preferences**
- **Content management, personalization, mobile**
- **Leaderboards, real-time analytics, caching**
- **Fraud detection, social networking, recommendation engine**
- **IoT applications, event tracking**
- **Systems of record, supply chain, health care, registrations, financial**
AWS: Purpose-built databases

- **Relational**
  - Amazon RDS
    - Aurora
    - Community
    - Commercial
  - MySQL
  - PostgreSQL
  - Oracle
  - SQL Server
  - MariaDB

- **Key-value**
  - Amazon DynamoDB
  - Amazon ElastiCache
    - Redis
    - Memcached

- **Document**
  - Amazon DocumentDB
  - Amazon Quantum Ledger Database

- **In-memory**
  - Amazon RDS
  - Amazon ElastiCache
  - Memcached

- **Graph**
  - Amazon Neptune

- **Search**
  - Amazon Elasticsearch Service

- **Time-series**
  - Amazon Timestream

- **Ledger**
  - Amazon Quantum Ledger Database
Exploring Each Database
Amazon Relational Database Service (RDS)

Managed relational database service with a choice of six popular database engines

- **Easy to administer**: No need for infrastructure provisioning, installing, and maintaining DB software
- **Available and durable**: Automatic Multi-AZ data replication; automated backup, snapshots, failover
- **Highly scalable**: Scale database compute and storage with a few clicks with no app downtime
- **Fast and secure**: SSD storage and guaranteed provisioned I/O; data encryption at rest and in transit
Traditional SQL

- TCP based wire protocol
- Well Known, lots of uses
- Common drivers (JDBC)
- Frequently used with ORMs
- Scale UP individual instances
- Scale OUT with read replicas
- Sharding at application level
- Lots of flavors but very similar language
- Joins

```sql
INSERT INTO table1
(id, first_name, last_name)
VALUES (1, 'Randall', 'Hunt');

SELECT col1, col2, col3
FROM table1
WHERE col4 = 1 AND col5 = 2
GROUP BY col1
HAVING count(*) > 1
ORDER BY col2
```
Amazon Aurora

MySQL and PostgreSQL-compatible relational database built for the cloud
Performance and availability of commercial-grade databases at 1/10th the cost

- Performance and scalability: 5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to 15 read replicas
- Availability and durability: Fault-tolerant, self-healing storage; six copies of data across three Availability Zones; continuous backup to Amazon S3
- Highly secure: Network isolation, encryption at rest/transit
- Fully managed: Managed by RDS: No hardware provisioning, software patching, setup, configuration, or backups

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
## SQL vs NoSQL

<table>
<thead>
<tr>
<th>SQL</th>
<th>NoSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimized for storage</td>
<td>Optimized for compute</td>
</tr>
<tr>
<td>Normalized/relational</td>
<td>Denormalized/hierarchical</td>
</tr>
<tr>
<td>Ad hoc queries</td>
<td>Instantiated views</td>
</tr>
<tr>
<td>Scale vertically</td>
<td>Scale horizontally</td>
</tr>
<tr>
<td>Good for OLAP</td>
<td>Built for OLTP at scale</td>
</tr>
</tbody>
</table>

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Amazon DynamoDB
Fast and flexible key value database service for any scale

Performance at scale
Consistent, single-digit millisecond response times at any scale; build applications with virtually unlimited throughput

Serverless
No server provisioning, software patching, or upgrades; scales up or down automatically; continuously backs up your data

Comprehensive security
Encrypts all data by default and fully integrates with AWS Identity and Access Management for robust security

Global database for global users and apps
Build global applications with fast access to local data by easily replicating tables across multiple AWS Regions
Amazon DynamoDB Data Structure

Table

Items

Attributes

Partition Key

Sort Key

Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities

All items for key
==, <, >, >=, <=
“begins with”
“between”
“contains”
“in”
sorted results
counts
top/bottom N values
DynamoDB Schema and Queries

- Connects over HTTP
- Global Secondary Indexes and Local Secondary Indexes
- Speed up queries with DAX
- Global tables (multi-region-multi-master)
- Transactions across multiple tables
- Change Streams
- Rich query language with expressions
- Provision read and write capacity units separately with or without autoscaling
- Also supports pay per request model

```python
import boto3
votes_table = boto3.resource('dynamodb').Table('votes')
resp = votes_table.update_item(
    Key={'name': editor},
    UpdateExpression="ADD votes :incr",
    ExpressionAttributeValues={":incr": 1},
    ReturnValues="ALL_NEW"
)
```
DynamoDB
Advancements over the last 22 months

February 2017
- Time To Live (TTL)

April 2017
- VPC endpoints

April 2017
- DynamoDB Accelerator (DAX)

June 2017
- Auto scaling

November 2017
- Global tables

November 2017
- On-demand backup

November 2017
- Encryption at rest

March 2018
- Point-in-time recovery

June 2018
- 99.999% SLA

August 2018
- Adaptive capacity

November 2018
- Transactions

November 2018
- On-demand
Amazon DocumentDB: Modern cloud-native architecture

What would you do to improve scalability and availability?

1. Decouple compute and storage
2. Distribute data in smaller partitions
3. Increase the replication of data (6x)
Amazon DocumentDB
Fast, scalable, and fully managed MongoDB-compatible database service

**Fast**
- Millions of requests per second with millisecond latency; twice the throughput of MongoDB

**Scalable**
- Separation of compute and storage enables both layers to scale independently; scale out to 15 read replicas in minutes

**Fully managed**
- Managed by AWS: no hardware provisioning; auto patching, quick setup, secure, and automatic backups

**MongoDB compatible**
- Compatible with MongoDB 3.6; use the same SDKs, tools, and applications with Amazon DocumentDB

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Document databases

- Data is stored in JSON-like documents
- Documents map naturally to how humans model data
- Flexible schema and indexing
- Expressive query language built for documents (ad hoc queries and aggregations)

**JSON documents are first-class objects of the database**

```json
{
    id: 1,
    name: "sue",
    age: 26,
    email: "sue@example.com",
    promotions: ["new user", "5%", "dog lover"],
    memberDate: 2018-2-22,
    shoppingCart: [
        {product: "abc", quantity: 2, cost: 19.99},
        {product: "edf", quantity: 3, cost: 2.99}
    ]
}
```
Amazon ElastiCache
Redis and Memcached compatible, in-memory data store and cache

- **Redis & Memcached compatible**
  - Fully compatible with open source Redis and Memcached

- **Extreme performance**
  - In-memory data store and cache for microsecond response times

- **Secure and reliable**
  - Network isolation, encryption at rest/transit, HIPAA, PCI, FedRAMP, multi AZ, and automatic failover

- **Easily scalable**
  - Scale writes and reads with sharding and replicas
Redis

- Redis Serialization Protocol over TCP (RESP)
- Supports Strings, Hashes, Lists, Sets, and Sorted Sets
- Simple commands for manipulating in memory data structures
- Pub/Sub features
- Supports clustering via partitions
Memcached

• Text and Binary protocols over TCP
• Small number of commands: set, add, replace, append, prepend, cas, get, gets, delete, incr, decr
• Client/Application based partitioning
Relationships enable new applications

Social networks

Restaurant recommendations

Retail fraud detection
Use cases for highly connected data

- Social networking
- Recommendations
- Knowledge graphs
- Fraud detection
- Life Sciences
- Network & IT operations
Different approaches for highly connected data

Purpose-built for a business process

Purpose-built to answer questions about relationships
Amazon Neptune
Fully managed graph database

Fast
Query billions of relationships with millisecond latency

Reliable
Six replicas of your data across three AZs with full backup and restore

Easy
Build powerful queries easily with Gremlin and SPARQL

Open
Supports Apache TinkerPop & W3C RDF graph models
# Leading Graph Models and Frameworks

**Property Graph**

- Open Source Apache TinkerPop™
- Gremlin Traversal Language

**Resource Description Framework (RDF)**

- W3C Standard
- SPARQL Query Language
Amazon Neptune high level architecture

TinkerPop / Gremlin

RDF / SPARQL

Amazon Neptune
High Performance Graph Engine (Durable, ACID with Immediate Consistency)

Cloud-native Storage Service

- Multi-AZ HA
- Read Replicas
- Encryption-at-rest

Bulk load from S3
Database Mgmt.
Gremlin

g.addV('person').property(id, 1).property('name', 'randall')
g.V('1').property(single, 'age', 27)
g.addV('person').property(id, 2).property('name', 'markus')
g.addE('knows').from(g.V('1')).to(g.V('2')).property('weight', 1.0)
g.V().hasLabel('person')
g.V().has('name', 'randall').out('knows').valueMap()

http://tinkerpop.apache.org/docs/current/reference/#graph-traversal-steps
SPARQL and RDF

Data:

```xml
```

Query:

```sparql
SELECT ?title
WHERE
{
}
```

This query, on the data above, has one solution:

Query Result:

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SPARQL Tutorial&quot;</td>
</tr>
</tbody>
</table>

https://www.w3.org/TR/sparql11-query/
THE BENEFIT OF URIs: LINKED DATA

Linking across datasets by referencing globally unique URIs

Example: PermlD (re)uses <http://sws.geonames.org/6252001/> as a global Identifier for the USA, which is an identifier rooted in GeoNames.
Amazon Timestream (sign up for the preview)

Fast, scalable, fully managed time-series database

- 1,000x faster and 1/10th the cost of relational databases
- Collect data at the rate of millions of inserts per second (10M/second)
- Trillions of daily events
- Adaptive query processing engine maintains steady, predictable performance
- Time-series analytics
- Built-in functions for interpolation, smoothing, and approximation
- Serverless
- Automated setup, configuration, server provisioning, software patching
Amazon Quantum Ledger Database (QLDB) (Preview)

Fully managed ledger database
Track and verify history of all changes made to your application’s data

- **Immutable**: Maintains a sequenced record of all changes to your data, which cannot be deleted or modified; you have the ability to query and analyze the full history.
- **Cryptographically verifiable**: Uses cryptography to generate a secure output file of your data’s history.
- **Highly scalable**: Executes 2–3X as many transactions than ledgers in common blockchain frameworks.
- **Easy to use**: Easy to use, letting you use familiar database capabilities like SQL APIs for querying the data.
AWS Database Migration Service (AWS DMS)

Migrating databases to AWS

Migrate between on-premises and AWS
Migrate between databases
Automated schema conversion
Data replication for zero-downtime migration

100,000+ databases migrated
AWS DMS—Logical replication

Start a replication instance

Connect to source and target databases

Select tables, schemas, or databases

Let the AWS Database Migration Service create tables and load data
Uses change data capture to keep them in sync
Switch applications over to the target at your convenience
How do we use these in Real Life?
Customers are moving to AWS Databases

Verizon is migrating over 1,000 business-critical applications and database backend systems to AWS, several of which also include the migration of production databases to Amazon Aurora.

By December 2018, Amazon.com will have migrated 88% of their Oracle DBs (and 97% of critical system DBs) moved to Amazon Aurora and Amazon DynamoDB. They also migrated their 50 PB Oracle Data Warehouse to AWS (Amazon S3, Amazon Redshift, and Amazon EMR).

Trimble migrated their Oracle databases to Amazon RDS and project they will pay about 1/4th of what they paid when managing their private infrastructure.

Wappa migrated from their Oracle database to Amazon Aurora and improved their reporting time per user by 75 percent.

Samsung Electronics migrated their Cassandra clusters to Amazon DynamoDB for their Samsung Cloud workload with 70% cost savings.

Intuit migrated from Microsoft SQL Server to Amazon Redshift to reduce data-processing timelines and get insights to decision makers faster and more frequently.

Equinox Fitness migrated its Teradata on-premises data warehouse to Amazon Redshift. They went from static reports to a modern data lake that delivers dynamic reports.

Eventbrite moved from Cloudera to Amazon EMR and were able to cut costs dramatically, spinning clusters up/down on-demand and using Spot (saving > 80%) and Reserved Instances.
Most enterprise database & analytics cloud customers

DOW JONES
NFL
Cognizant
21ST CENTURY FOX
Adobe
Adobe
Cerner
ticketmaster
Expedia
REDFIN
HBO
Samsung

NASDAQ
Blackboard
FINRA
Sysco
SEGA
Vodafone
Comcast

COMCAST
Telemx
Unilever
Canon
Intuit
Atlassian
ANA
Broadsoft

Nike

Mlbam
Accenture
LG
Siemens
Liberty Mutual
Sharp
Mobil

Gogo

New York Public Library

British Gas

American Gas

Rackspace

Mashable

Jenny Craig

Merck

FT

Hilton

Midas

Sonoma

Shutterstock

Neumours

Hearst

Societe Generale

Morgan Stanley

Formula 1

Nikkei

Scripps

Seattle Times

University of Chicago

Outback

Nike

Trend 

HBX

FCB Barcelona

Heineken

Pepsi

Virgin

Bristol Myers Squibb

Hulu

Cal Poly

FICO

CSS Interactive

Zendesk

Mitel

Dow Jones

Nielsen

Amazon

Experian

SRA OSS

McDonald

TRIMBLE

Zillow

Kimberly-Clark

Eventbrite

Loggly

Illumina

ASU

LoginWorks

Sierra-Cedar

Netflix

Netflix

Takeda

Hudl

JustGiving

SK

Planet

Boingo


© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Most startup database & analytics cloud customers
Markus Ostertag
CEO @ Team Internet AG
Problem: Matching IPs to Ranges in Real-Time Bidding

Real-Time Bidding Requirements:
• High throughput (>25000 req/s)
• Response time is critical
• Scalability
## Possible choices

<table>
<thead>
<tr>
<th></th>
<th>Relational</th>
<th>Key-value</th>
<th>In-memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upside</strong></td>
<td>Well known, Flexible Indices, Query language, Data scheme obvious</td>
<td>High throughput, low-latency reads, endless scale</td>
<td>Query by key with microsecond latency</td>
</tr>
</tbody>
</table>
### Possible choices

<table>
<thead>
<tr>
<th></th>
<th>Relational</th>
<th>Key-value</th>
<th>In-memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upside</strong></td>
<td>Well known, Flexible Indices, Query language, Data scheme obvious</td>
<td>High throughput, low-latency reads, endless scale</td>
<td>Query by key with microsecond latency</td>
</tr>
</tbody>
</table>
DynamoDB?

But how?

- Sharding and Index downsides addressed by transforming data during ingest
- Accept multiple entries per range -> sharding & query possible/optimized

<table>
<thead>
<tr>
<th>Partition key</th>
<th>Sort key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_net</td>
<td>ip_end</td>
</tr>
<tr>
<td>18.208</td>
<td>315686911</td>
</tr>
<tr>
<td>18.209</td>
<td>315752447</td>
</tr>
<tr>
<td>52.194</td>
<td>885194751</td>
</tr>
<tr>
<td>52.195</td>
<td>885260287</td>
</tr>
<tr>
<td>54.155</td>
<td>916193279</td>
</tr>
</tbody>
</table>
DynamoDB it is!

```javascript
var _findIp = function(ip, ipLong, dynamo, callback) {
  var ip_parts = ip.split(".");
  var net_string = ip_parts[0]+"."+ip_parts[1];
  var params = {
    TableName: "ip-ranges",
    KeyConditionExpression: "ip_net = :net and ip_end >= :ipLong",
    FilterExpression: "ip_start <= :ipLong",
    ExpressionAttributeValues: {
      ":net": {S: net_string},
      ":ipLong": {N: ipLong.toString()}
    }
  };
  dynamo.query(params, callback);
};
```
DynamoDB it is!

```javascript
var _findIp = function(ip, ipLong, dynamo, callback) {
  var ip_parts = ip.split('.');
  var net_string = ip_parts[0] + '.' + ip_parts[1];

  var params = {
    TableName: "ip-ranges",
    KeyConditionExpression: "ip_net = :net and ip_end >= :ipLong",
    FilterExpression: "ip_start <= :ipLong",
    ExpressionAttributeValues: {
      ':net': {S: net_string},
      ':ipLong': {N: ipLong.toString()}
    }
  };
  dynamo.query(params, callback);
};
```
DynamoDB it is!

```javascript
var _findIp = function(ip, ipLong, dynamo, callback) {
    var ip_parts = ip.split(".");
    var net_string = ip_parts[0]+"."+ip_parts[1];
    var params = {
        TableName: "ip-ranges",
        KeyConditionExpression:
            "ip_net = :net and ip_end >= :ipLong",
        FilterExpression: "ip_start <= :ipLong",
        ExpressionAttributeValues: {
            ":net": {S: net_string},
            ":ipLong": {N: ipLong.toString()}
        }
    };
    dynamo.query(params, callback);
};
```
Learnings

• Focus on “core requirements”
• Be creative about data schemes
• Accept minor downsides – very often there is no perfect solution
• Use upsides of chosen database to optimize (i.e. adding DAX for caching)
Knowledge Graphs @ Siemens

Sebastian Brandt
Steffen Lamparter

Semantics and Reasoning Group
Siemens Corporate Technology

February 2019
Knowledge graphs become especially powerful for managing complex queries and heterogeneous data

Why Knowledge Graphs?

• **Graphs are a natural way** to represent entities and their relationships
• **Graphs can capture a broad spectrum of data** (structured / unstructured)
• **Graphs can be managed efficiently**

- **Game-changing data integration**
- **Robust data quality assurance**
- **Intuitive domain modelling**
- **Flexibility & performance**
- **Low up-front investment**
What are Graphs? Knowledge representation formalism semantic descriptions of entities and their relationships

Rules make it possible to add further expert knowledge, e.g. "Siemens has to be a company, as a person is working there"

Objects
Real-world objects (things, places, people) and abstract concepts (genres, religions, professions)

Relationships
Logical connection between two objects e.g. Joe Kaeser is born in Arnbruck

Semantic descriptions
The semantic description indicates the meaning of an object or relation, e.g. Joe Kaeser is a person
Knowledge graphs become a powerful addition to traditional data warehouses for managing heterogeneous data with complex relations

<table>
<thead>
<tr>
<th>Dimensions of data relations</th>
<th>Traditional data warehouses</th>
<th>Knowledge graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limited relations across data, e.g. time series data</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>High number &amp; complex relations, e.g. social networks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variety of questions to be answered</th>
<th>Traditional data warehouses</th>
<th>Knowledge graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;We have a clear scope of user questions&quot; → fixed set of dashboard needed</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>&quot;We do not yet know all the different questions, users will ask&quot; → Chatbot</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data heterogeneity</th>
<th>Traditional data warehouses</th>
<th>Knowledge graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only few different data types and data sources e.g. invoicing database in ERP</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>Analysis needs to combine heterogeneous data, e.g. sensor data, text data, product data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality of data</th>
<th>Traditional data warehouses</th>
<th>Knowledge graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensured consistency of stored data</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>Need to handle imperfect, incomplete or inconsistent data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Use cases for knowledge graphs can be clustered into five categories – overview and use case examples

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data quality</td>
<td>Improving data availability and quality by combining and comparing data from various sources to fill in missing data sets or identify potentially wrong data and data duplicates</td>
</tr>
<tr>
<td>Data access &amp; dashboarding</td>
<td>Maintaining up-to-date meta-data, creating transparency on all available data and making them accessible to users via queries</td>
</tr>
<tr>
<td>Digital companion</td>
<td>Enhancing features of existing products or services with digital companions that are able to understand and process user questions and providing the needed data insights</td>
</tr>
<tr>
<td>Recommender system</td>
<td>Providing users high quality recommendations by identifying similarities in historical data</td>
</tr>
<tr>
<td>Constraints &amp; planning</td>
<td>Enabling autonomous systems to understand data and its dependencies and take own decisions, such as autonomous planning of production processes</td>
</tr>
</tbody>
</table>

Degree of complexity
Example: gas-turbine maintenance planning

- Cross-life-cycle data integration at PS DO: gas-turbines, including maintenance, repair, monitoring, and configuration data
- Holistic engineering-centric domain model
- Intuitive graph queries independent of source data schemata
- Virtual integration of time-series data
A semantic data model enables flexible linking and an integrated, intuitive API for applications.
Creating perfect places based on Services – a user-centric holistic approach to the modern workplace ...

<table>
<thead>
<tr>
<th>Customer Interest</th>
<th>Relevant KPI's</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy and asset efficiency</td>
<td>€ Optimizing CAPEX and OPEX</td>
</tr>
<tr>
<td></td>
<td>€ CO₂ emissions</td>
</tr>
<tr>
<td></td>
<td>€ Asset Performance/Useful Life</td>
</tr>
<tr>
<td>Space efficiency</td>
<td>€ Cost per space unit</td>
</tr>
<tr>
<td></td>
<td>€ Workplace Utilization</td>
</tr>
<tr>
<td></td>
<td>€ Revenue per space unit</td>
</tr>
<tr>
<td></td>
<td>€ Vacancy Rate</td>
</tr>
<tr>
<td>Individual efficiency and comfort</td>
<td>€ Employee productivity</td>
</tr>
<tr>
<td></td>
<td>€ Employee satisfaction</td>
</tr>
</tbody>
</table>
## Industrializing Knowledge Graphs

### Industrial Knowledge Graph

<table>
<thead>
<tr>
<th>Decision Making</th>
<th>Storage and Integration</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning and Constraint Solving, Machine/Deep Learning, Question Answering</td>
<td>Graph/NoSQL databases, Constraints and Rules, Probabilistic programming, Ontologies</td>
<td>NLP/Text understanding, Machine/Deep Learning, Computer vision, Sound recognition, Virtual data Integration, Information retrieval</td>
</tr>
</tbody>
</table>

### Relevant technologies

#### Decision Making
- Reasoning and Constraint Solving
- Machine/Deep Learning
- Question Answering

#### Storage and Integration
- Graph/NoSQL databases
- Constraints and Rules
- Probabilistic programming
- Ontologies

#### Generation
- NLP/Text understanding
- Machine/Deep Learning
- Computer vision
- Sound recognition
- Virtual data Integration
- Information retrieval

### R&D Areas

#### Decision Making
- Explanation of AI decisions
- Data access: Semantic Search
- Machine Learning on Graphs for recommendations, quality, etc.

#### Storage and Integration
- Reusable Semantic Modelling and Knowledge Graphs
- Data integration and cleaning (e.g., entity reconciliation)

#### Generation
- Extraction from unstructured data (inclusive text, audio, image)
- Automatic semantic annotation of structured data
- Learning of domain-specific rules/patterns

---

**Humans**

**Machines**
Dr. Sebastian Brandt
Senior Key Expert – Knowledge Graph and Data Management
CT RDA BAM SMR-DE

Dr. Steffen Lamparter
Head of Research Group Semantics & Reasoning
CT RDA BAM SMR-DE
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 München Germany

E-mail
steffen.lamparter@siemens.com

Intranet
intranet.siemens.com/ct
Thank you!
Please complete the session survey.