Before we begin

```
git clone git@github.com:qa/pv243-a4m36jee-2016-infinispan-
    seminar-autumn.git
cd pv243-a4m36jee-2016-infinispan-seminar-autumn
git checkout task1
mvn clean package
mvn wildfly:run
```

Optionally:

```
wget http://downloads.jboss.org/infinispan/9.2.1.Final/
    infinispan-server-9.2.1.Final-bin.zip
```

Infinispan

Vojtěch Juránek

JBoss - a division by Red Hat

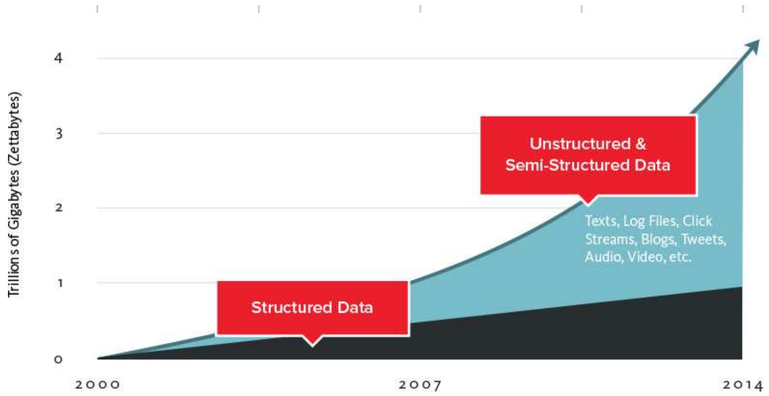
12. 4. 2018, FI MUNI, Brno

Course materials download

- Course materials, including this presentation:
  `https://developer.jboss.org/wiki/AdvancedJavaEELabFIMUNIJaro2018/`
- This presentation (and the source code):
  `https://github.com/vjuranek/presentations/tree/master/MUNI_Brno2018_spring`

Data today

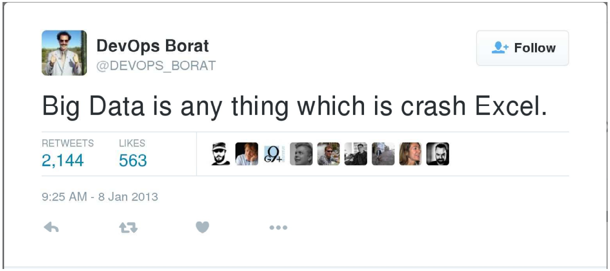
Data today



Source: `http://www.couchbase.com/nosql-resources/what-is-no-sql`

How big is Big data?

How big is Big data?


DevOps Borat
@DEVOPS_BORAT

Follow

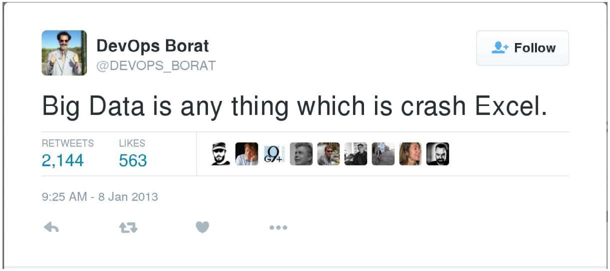
Big Data is any thing which is crash Excel.

RETWEETS 2,144 LIKES 563

9:25 AM - 8 Jan 2013

Source: `https://twitter.com/DEVOPS_BORAT/status/288698056470315008`

How big is Big data?


DevOps Borat
@DEVOPS_BORAT

Big Data is any thing which is crash Excel.

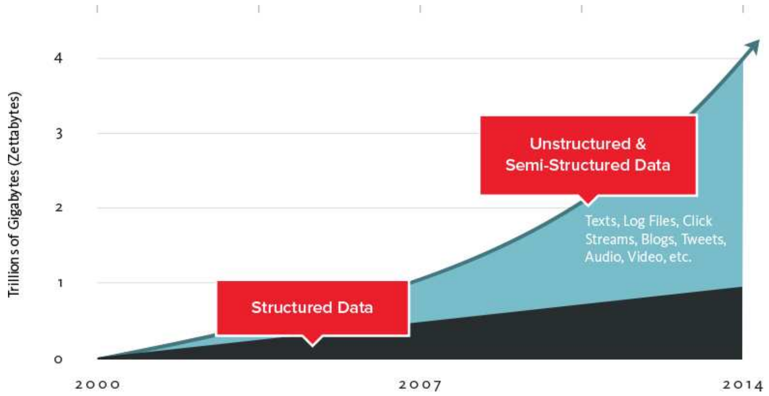
RETWEETS 2,144 LIKES 563

9:25 AM - 8 Jan 2013

Source: `https://twitter.com/DEVOPS_BORAT/status/288698056470315008`

- Data collection so large and complex it’s impossible to process it on one computer
- You can scale up, but sooner or later you’ll have to scale out

Structure of the data



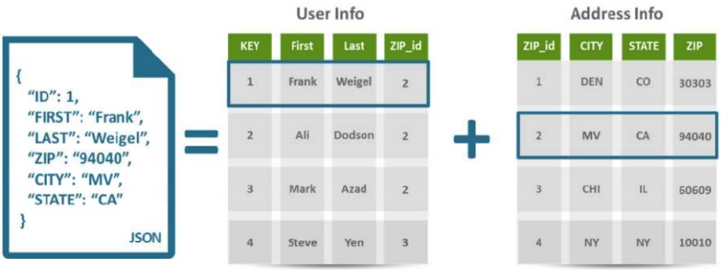
Source: `http://www.couchbase.com/nosql-resources/what-is-no-sql`

Big data - some of the challenges

- Analysis run on top of the huge amount of data
- Ability to store huge amount of unstructured data (often for performance reasons)
- But also ability to talk to RDBMS or query structured data is often needed as well
- Highly scalable solution (also because of cost effectiveness)
- Cloud architecture - everything is ephemeral
- Information privacy

NoSQL

- Nature of the data
- More flexible data mode
- Better scalablity
- Performance



```
{
  "ID": 1,
  "FIRST": "Frank",
  "LAST": "Weigel",
  "ZIP": "94040",
  "CITY": "MV",
  "STATE": "CA"
}
```

User Info

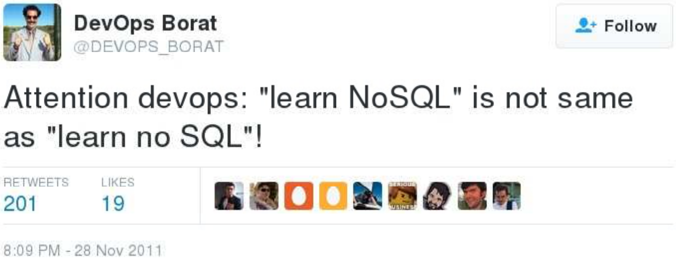
KEY	First	Last	ZIP_id
1	Frank	Weigel	2
2	Ali	Dodson	2
3	Mark	Azad	2
4	Steve	Yen	3

Address Info

ZIP_id	CITY	STATE	ZIP
1	DEN	CO	30303
2	MV	CA	94040
3	CHI	IL	60609
4	NY	NY	10010

Source: `www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf`

NoSQL



DevOps Borat
@DEVOPS_BORAT

Follow

Attention devops: "learn NoSQL" is not same as "learn no SQL"!

RETWEETS 201 LIKES 19

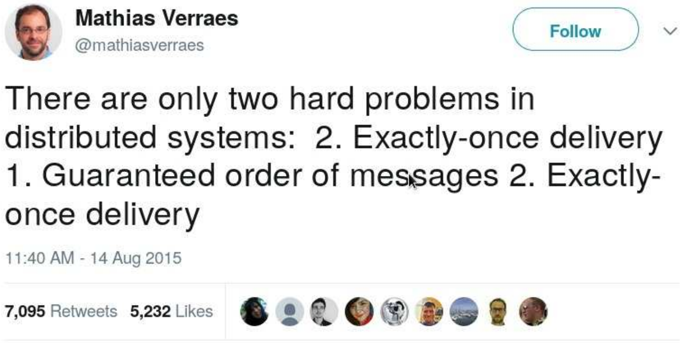
8:09 PM - 28 Nov 2011

Source: `https://twitter.com/devops_borat/status/141368065110708224`

Distributed databases

- Good scalability
- Better perfomance
- But also bring a lot of different challenges

Distributed systems


Mathias Verraes
@mathiasverraes

There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

11:40 AM - 14 Aug 2015

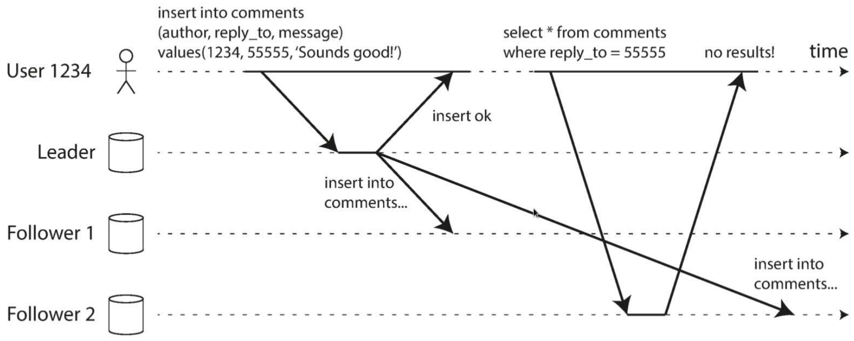
7,095 Retweets 5,232 Likes

Source: `https://twitter.com/mathiasverraes/status/632260618599403520`

More related jokes on
`https://martinfowler.com/bliki/TwoHardThings.html`

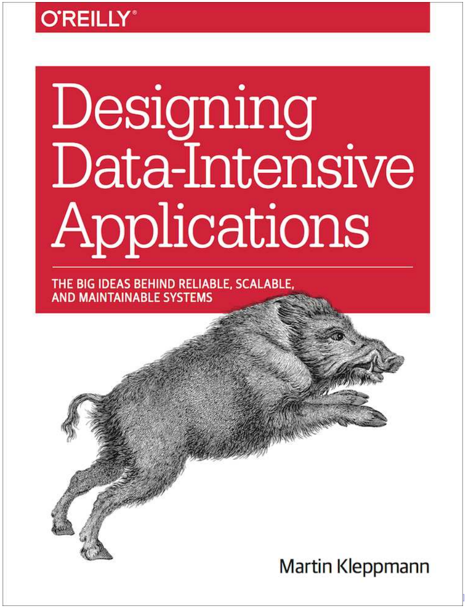
Distributed databases

You probably don't want to implement it yourselves, just once example (read after write/read your writes consistency):



Source: M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, Inc., 2016

Distributed databases and data intensive applications



O'REILLY®

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

Martin Kleppmann

What is a in-memory data grid?

What is a data grid?

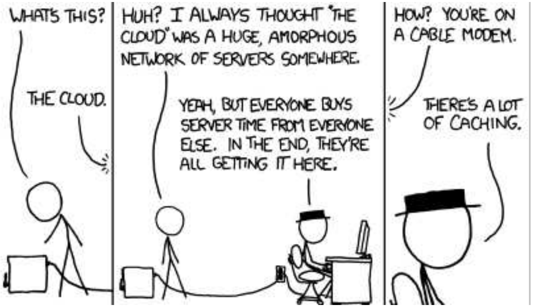
In-memory = all data is kept in memory

Grid = too big to kept data on one node, data is distributed across more/many nodes

What is a data grid?

In-memory = all data is kept in memory
Grid = too big to kept data on one node, data is distributed across more/many nodes

- An in-memory distributed data store designed for fast access to large volumes of data and scalability.
- Commonly a complementary layer to the relational database or some other persistent data storage, usually as a cache:


WHAT'S THIS?
THE CLOUD.
HUH? I ALWAYS THOUGHT "THE CLOUD" WAS A HUGE, AMORPHOUS NETWORK OF SERVERS SOMEWHERE.
YEAH, BUT EVERYONE BUYS SERVER TIME FROM EVERYONE ELSE. IN THE END, THEY'RE ALL GETTING IT HERE.
HOW? YOU'RE ON A CABLE MODEM.
THERE'S A LOT OF CACHING.

Source: Part of xkcd #908

Key data grid characteristics

- In-memory, distributed caching
- Elastic and scalable
- Advanced querying
- Data replication
- Processing for streaming data
- Transaction capabilities

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Infinispan

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

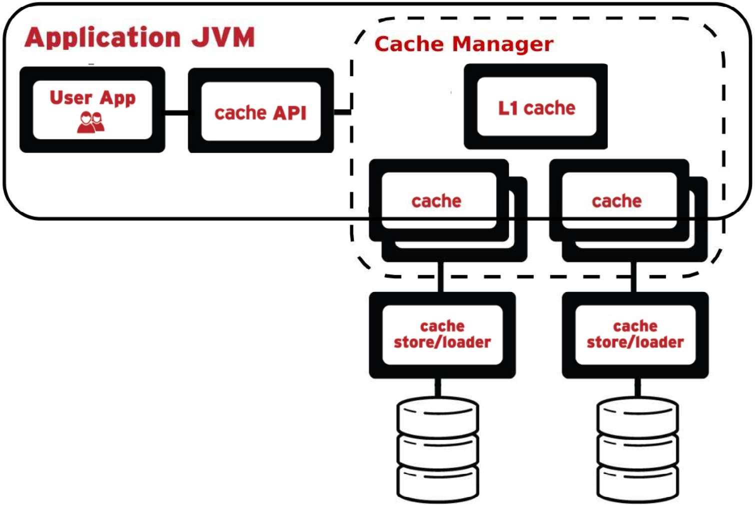
Infinispan cache

- Infinispan takes care about all that hard stuff.
- From user perspective Infinispan cache is **just a map!**

```
    DefaultCacheManager cacheManager = new DefaultCacheManager(
        "my_ispn_config.xml");
    Cache<String, String> cache = cacheManager.getCache("
        myCache");

    cache.put("key", "value");
    String value = cache.get("key");
```

- ISPN configuration can be either programmatic (preferred for demos) or via XML (preferred in production as you don't have to re-compile the code due to conf. changes).

Infinispan (embedded) high level architecture



Basic features: eviction

Removing entries from the cache: eviction

```
ConfigurationBuilder().eviction().size(5).strategy(
    EvictionStrategy.LRU)
```

```
Configuration conf = new ConfigurationBuilder().eviction().size
    (5).strategy(EvictionStrategy.LIRS).build();
EmbeddedCacheManager ecm = new DefaultCacheManager(conf);
Cache<String, String> cache = ecm.getCache();

for (int i = 0; i < 100; i++) {
    cache.put("key" + i, "value" + i);
}

System.out.printf("Cache size: %d\n", cache.size());
ecm.stop();
```

Basic features: cache listener

```
cache.addListener(new EntryCreatedListener());
```

- There are actually two events emitted, before given operation happens and once it's finished.
- You can distinguish them by calling `isPre()` on the event (`true` for events prior the operation)

```java
@Listener
public class EntryCreatedListener {
    @CacheEntryCreated
    public void onCreated(CacheEntryCreatedEvent e) {
        if (e.isPre()) {
            System.out.printf("Created %s -> %s\n", e.getKey(),
                e.getValue());
        }
    }
}
```

```java
EmbeddedCacheManager cm = new DefaultCacheManager();
Cache<String, String> cache = cm.getCache();
cache.addListener(new EntryCreatedListener());

for (int i = 0; i < 100; i++) {
    cache.put("key" + i, "value" + i);
}
cm.stop();
```

Basic features: CDI

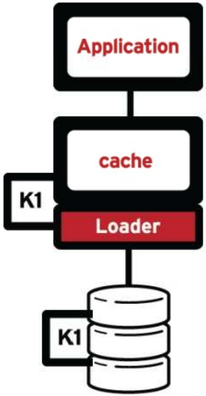
```java
@Qualifier
@Target({ElementType.FIELD, ElementType.PARAMETER, ElementType.
    METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface EvictionCache {
}
```

```java
@ConfigureCache("testcache")
@EvictionCache
@Produces
public Configuration greetingCacheConfiguration() {
    return new ConfigurationBuilder().eviction().strategy(
        EvictionStrategy.LRU).size(5).build();
}
```

```java
@Inject
@EvictionCache
private Cache<String, String> cache;
```

Persistence: Cache stores

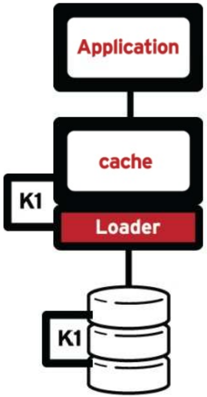
A way how to store cache content in some external (persistent) storage.



Persistence: Cache stores

A way how to store cache content in some external (persistent) storage.
Two modes:

- Synchronous (write-through)
- Asynchronous (write-behind)



Persistence: Cache stores

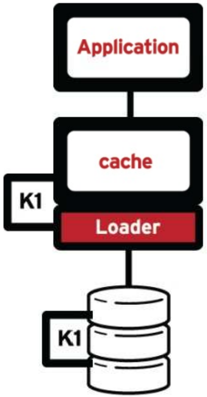
A way how to store cache content in some external (persistent) storage.
Two modes:

- Synchronous (write-through)
- Asynchronous (write-behind)

Cache stores:

- Single file store and soft-index file store
- JDBC and JPA cache stores
- LevelDB cache store
- Cloud cache store
- Remote store
- Cassandra store
- . . . and others

Also possible to define custom cache store.



Persistence: file cache store example

```
cfg.persistence().addSingleFileStore().location("/tmp/ispn-
    store");
```

```
ConfigurationBuilder cfg = new ConfigurationBuilder();
cfg.persistence().addSingleFileStore();
DefaultCacheManager cm = new DefaultCacheManager(cfg.build());
Cache<String, String> cache = cm.getCache("test");

for (int i = 0; i < 100; i++) {
    cache.put("key" + i, "value" + i);
}
System.out.printf("Cache size: %d\n", cache.size());

cache.stop();
cache.start();

System.out.printf("Cache size: %d\n", cache.size());
cm.stop();
```

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn’t support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Examples: querying

```java
public class Person {
  String name;
  String surname;

  public Person(String name, String surname) {
    this.name = name;
    this.surname = surname;
  }
}
```

Examples: querying

```java
ConfigurationBuilder cb = new ConfigurationBuilder();
EmbeddedCacheManager cm = new DefaultCacheManager(cb.build());
Cache<String, Person> cache = cm.getCache();
cache.put("person1", new Person("Will", "Shakespeare"));

// Obtain a query factory for the cache
QueryFactory<?> queryFactory = Search.getQueryFactory(cache);

// Construct a query
Query query = queryFactory.from(Person.class).having("name").eq
    ("Will").toBuilder().build();

// Execute the query
List<Person> matches = query.list();

matches.forEach(person -> System.out.printf("Match: %s", person
    ));
cm.stop();
```

Transactions, consistency, locking and isolation

- JTA-compliant transactions
- Deadlock detection and recovery (e.g. when ISPN fails during commit phase of the transaction)
- Data versioning
- Ensures consistency of data, consistency guarantee: lock for key `K` is always acquired on the same node of the cluster (key **primary owner**), regardless of where the transaction originates

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

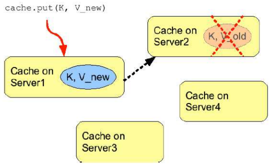
Clustering modes

- Under the hood leverages JGroups project for clustering.
- Data is distributed and replicated in the background.
- Nodes can be added or removed smoothly.

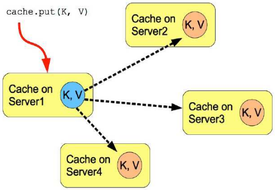
Clustering modes

- Under the hood leverages JGroups project for clustering.
- Data is distributed and replicated in the background.
- Nodes can be added or removed smoothly.

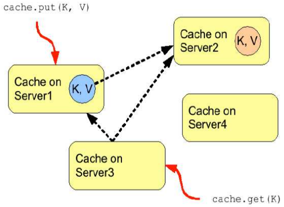
- Local - no clustering
- Invalidation



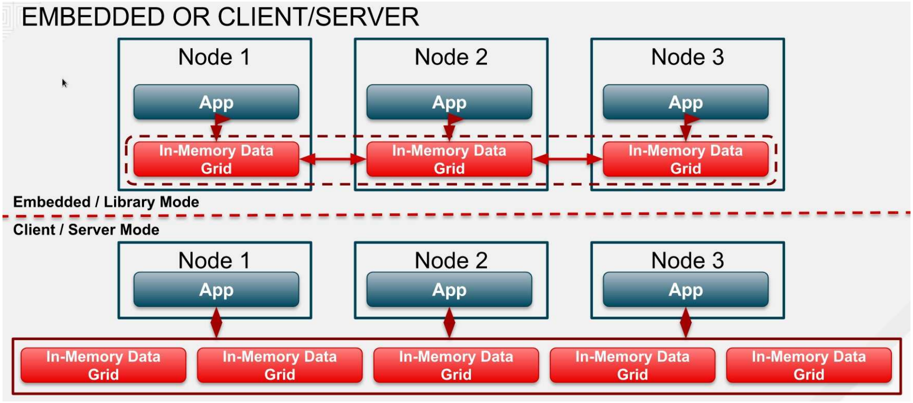
- Replicated



- Distributed

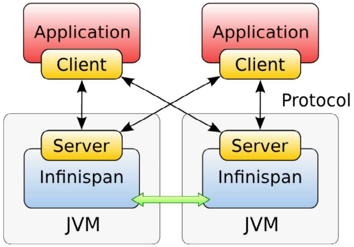


Infinispan modes



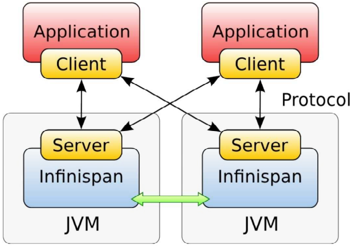
Remote protocols

- Hot Rod
  - hashing and topology aware
  - failover during topology changes
  - smart request routing
- Memcached
- REST



Remote protocols

- Hot Rod
  - hashing and topology aware
  - failover during topology changes
  - smart request routing
- Memcached
- REST



Protocol	Format	Client libs	Clustered	Smart routing	Load balancing / Failover
Hot Rod	binary	Java, C++, C#, JS	yes	yes	dynamic
Memcached	text	many	yes	no	only predefined server list
REST	text	any HTTP client	yes	no	any HTTP load balancer

Hot Rod clients

Compatible with Java and non-Java platforms. Based on Protocol Buffers - Google's data interchange format.

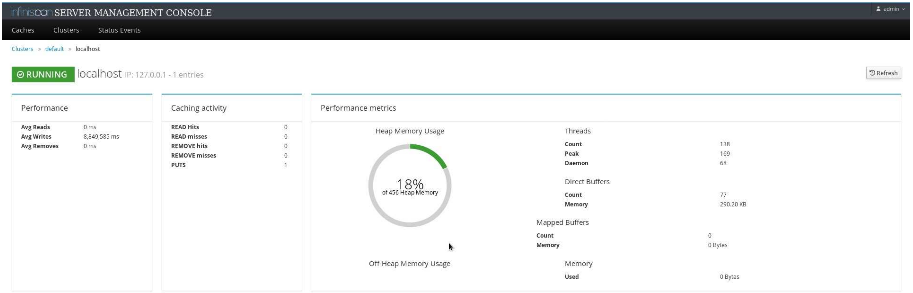
Clients for

- Java
- C#
- C++
- JavaScript
- Python
- Ruby

Python and Ruby clients have only basic functionality.

Management console

- Easy configuration
- Monitoring and statistics



Performance	
Avg Reads	0 ms
Avg Writes	8,849,585 ms
Avg Removes	0 ms

Caching activity	
READ Hits	0
READ misses	0
REMOVE hits	0
REMOVE misses	0
PUTS	1

Threads	
Count	138
Peak	169
Daemon	68

Direct Buffers	
Count	77
Memory	290.20 KB

Mapped Buffers	
Count	0
Memory	0 Bytes

Memory	
Used	0 Bytes

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Some other features - brief and selective list

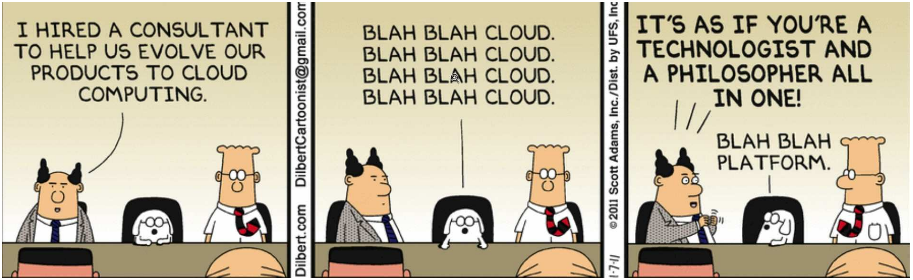
- Full JSR-107 support (Java Temporary Caching API)
- Advanced security feature (role based access, encryption, integration with LDAP, Kerberos etc.)
- Remote events
- Continuous query
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors
- Distributed streams

Examples of usecases

- Cache for backend
- Fast data backend
- HTTP session off-loading
- Hibernate 2-nd level cache
- In-memory Lucene index
- Fast data backend for Apache Spark or Hadoop
- . . .

OpenShift

- Try Infinispan on OpenShift
- https://www.openshift.com/
- Platform as a Service (PaaS)


I HIRED A CONSULTANT TO HELP US EVOLVE OUR PRODUCTS TO CLOUD COMPUTING.
BLAH BLAH CLOUD. BLAH BLAH CLOUD. BLAH BLAH CLOUD. BLAH BLAH CLOUD.
IT'S AS IF YOU'RE A TECHNOLOGIST AND A PHILOSOPHER ALL IN ONE!
BLAH BLAH PLATFORM.
Dilbert.com DilbertCartoonist@gmail.com
©2011 Scott Adams, Inc./Dist. by UFS, Inc
1-7-11

Source: http://dilbert.com/strip/2011-01-07

Search OpenShift demos on
https://github.com/infinispan-demos/links

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Materials from this course

- This presentation: https://github.com/vjuranek/presentations/tree/master/CTU_Prague2016_fall
- ISPN embedded tutorial (The Weather App): http://infinispan.org/tutorials/embedded
- GitHub repo: https://github.com/infinispan/infinispan-embedded-tutorial
- ISPN simple tutorials: https://github.com/infinispan/infinispan-simple-tutorials
- ISPN qickstarts (simple applications) at the bottom of the page: http://infinispan.org/tutorials
- Some more ISPN snippets: https://github.com/vjuranek/infinispan-snippets

Infinispan downloads:

- Main ISPN download page: http://infinispan.org/download/
- If you want to play with ISPN in Docker: https://hub.docker.com/r/jboss/infinispan-server/

Further study materials

- Infinispan documentation
- JSR 107: JCACHE - Java Temporary Caching API
- M. Surtani, F. Marchioni, Infinispan Data Grid Platform, Packt Publishing, 2012
- W. dos Santos, Infinispan Data Grid Platform Definitive Guide, Packt Publishing, 2015
- M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, Inc., 2016
- M. Takada, Distributed systems for fun and profit

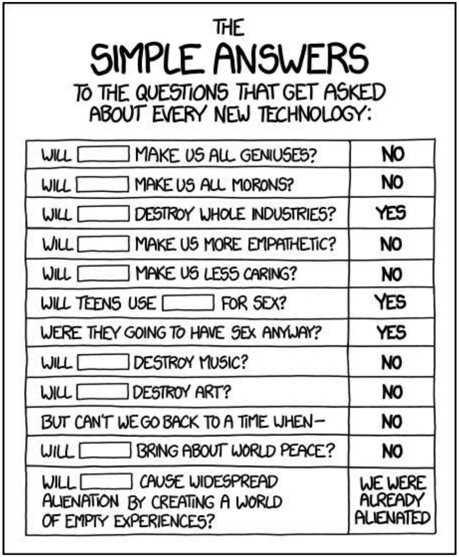
- B. Burke, A. Rubinger, Enterprise JavaBeans 3.1, 6th Edition, O'Reilly Media, Inc., 2010

- Coursera: Cloud Computing Concepts
- Coursera: Cloud Computing Concepts: Part 2
- Coursera: Cloud Computing Applications

Student projects/theses with Infinispan

- `https://developer.jboss.org/wiki/StudentContributorProjectsWithInfinispan`
- `https://diplomky.redhat.com/`
- Inerested to work with Infinispan but non of the theses is interesting for you - drop me an email on `vjuranek[at]redhat.com`, we try to figure out something.

Question?



THE SIMPLE ANSWERS TO THE QUESTIONS THAT GET ASKED ABOUT EVERY NEW TECHNOLOGY:

WILL ☐ MAKE US ALL GENIUSES?	NO
WILL ☐ MAKE US ALL MORONS?	NO
WILL ☐ DESTROY WHOLE INDUSTRIES?	YES
WILL ☐ MAKE US MORE EMPATHETIC?	NO
WILL ☐ MAKE US LESS CARING?	NO
WILL TEENS USE ☐ FOR SEX?	YES
WERE THEY GOING TO HAVE SEX ANYWAY?	YES
WILL ☐ DESTROY MUSIC?	NO
WILL ☐ DESTROY ART?	NO
BUT CAN'T WE GO BACK TO A TIME WHEN—	NO
WILL ☐ BRING ABOUT WORLD PEACE?	NO
WILL ☐ CAUSE WIDESPREAD ALIENATION BY CREATING A WORLD OF EMPTY EXPERIENCES?	WE WERE ALREADY ALIENATED

Source: `https://xkcd.com/1289/`

http://infinispan.org/

Thank you for your attention!

Backup slides

Infinispan embedded tutorial

Simple weather app using embedded Infinispan

- http://infinispan.org/tutorials/embedded/
- https://github.com/infinispan/infinispan-embedded-tutorial

```
git clone https://github.com/infinispan/infinispan-embedded
    -tutorial.git
cd infinispan-embedded-tutorial
git checkout -f step-2
sed -i 's/<!-- a/<a/;s/t -->/t>/' pom.xml #switch to local
    random weather service
mvn clean package
mvn exec:exec
```

Basic features: expiration

Removing entries from the cache: expiration

```
ConfigurationBuilder().expiration().maxIdle(5000L)
```

```
Configuration conf = new ConfigurationBuilder().expiration().
    maxIdle(expiration).build();
EmbeddedCacheManager ecm = new DefaultCacheManager(conf);
Cache<String, String> cache = ecm.getCache();

for (int i = 0; i < 100; i++) {
    cache.put("key" + i, "value" + i);
}

System.out.printf("Cache size: %d\n", cache.size());
Thread.sleep(expiration);
System.out.printf("Cache size: %d\n", cache.size());

ecm.stop();
```

Examples: querying with index

```
@Indexed
public class Person {
  @Field(analyze = Analyze.NO)
  String name;

  @Field(analyze = Analyze.NO, indexNullAs = Field.
      DEFAULT_NULL_TOKEN)
  String surname;

  public Person(String name, String surname) {
    this.name = name;
    this.surname = surname;
  }
}
```

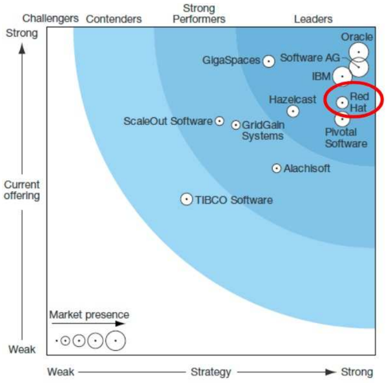
Examples: querying with index

```
ConfigurationBuilder cb = new ConfigurationBuilder();
cb.indexing().index(Index.ALL); //.addProperty("default.
    directory_provider", "ram");
EmbeddedCacheManager cm = new DefaultCacheManager(cb.build());
Cache<String, Person> cache = cm.getCache();
cache.put("person1", new Person("Will", "Shakespeare"));

// Obtain a query factory for the cache
QueryFactory<?> queryFactory = Search.getQueryFactory(cache);

// Construct a query
Query query = queryFactory.from(Person.class).having("name").eq
    ("Will").toBuilder().build();

// Execute the query
List<Person> matches = query.list();

matches.forEach(person -> System.out.printf("Match: %s", person
    ));
cm.stop();
```

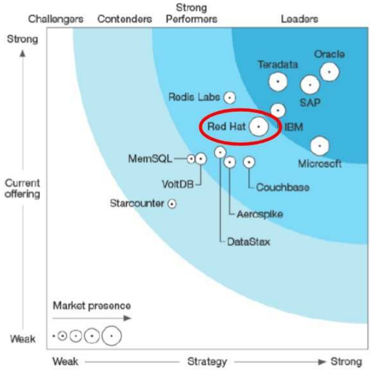
Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Analyst recognition



Forrester Wave™: In-Memory Data Grids, Q3 2015



Forrester Wave™: In-Memory Databases, Q1 2017

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.

1. Thread1: tx.begin()
2. Thread1: cache.get(k) returns v
3. Thread2: tx.begin()
4. Thread2: cache.get(k) returns v
5. Thread2: cache.put(k, v2)
6. Thread2: tx.commit()
7. Thread1: cache.get(k)

With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.

1. Thread1: tx.begin()
2. Thread1: cache.get(k) returns v
3. Thread2: tx.begin()
4. Thread2: cache.get(k) returns v
5. Thread2: cache.put(k, v2)
6. Thread2: tx.commit()
7. Thread1: cache.get(k)

With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
    - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
    - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
    1. Thread1: tx.begin()
    2. Thread1: cache.get(k) returns v
    3. Thread2: tx.begin()
    4. Thread2: cache.get(k) returns v
    5. Thread2: cache.put(k, v2)
    6. Thread2: tx.commit()
    7. Thread1: cache.get(k)

    With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Features in Infinispan 8

- Functional API
- Distributed streams
- Continuous querying, grouping and aggregation
- New management console
- Integration with Apache Spark and Hadoop
- . . . and more

Commercial break: Protocol Buffers

Protocol Buffers (protobuf) are language-neutral, platform-neutral, extensible mechanism for serializing structured data developed by Google.

- Supports C++, C#, Go, Java, Python.
- You need to define data structure in protobuf file.
- In ISPN you can use also annotations in the your model.

Example of protobuf file:

```
message Address {
  required string street = 1;
  required string postCode = 2;
}

 message Person {
  optional int32 id = 1;
  required string name = 2;
  required string surname = 3;
  optional Address address = 4;
  optional string license = 5;
  enum Gender {
    MALE = 0;
    FEMALE = 1;
  }
}
```