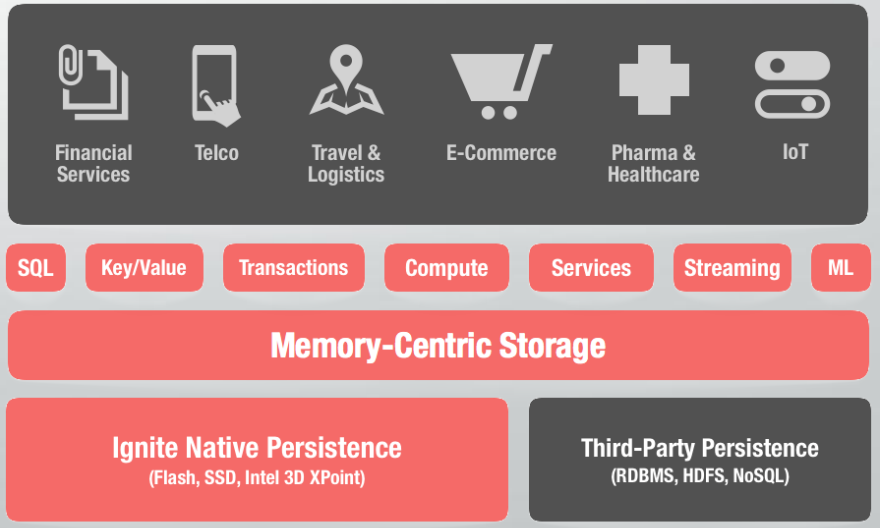
Programowanie w chmurze na platformie Java EE

Wykład 4 - dr inż. Piotr Zając

Apache Ignite

- Ignite™ is a memory-centric distributed database, caching, and processing platform for transactional, analytical, and streaming workloads delivering in-memory speeds at petabyte scale
- Ignite is Used by ING, Sberbank, HomeAway, Wellington, American Airlines, Yahoo! Japan, 24 Hour Fitness, JacTravel, and many more
- Documentation: https://apacheignite.readme.io/docs
- Read Ignite Facts: https://apacheignite.readme.io/docs/ignite-facts



Characteristics

- All nodes have the same role
- One physical machine is not limited to one Ignite node instance
- There is no Management Node - preventing from single point of failure
- By default data is stored in distributed structures in memory
- Default behavior is to work as In-Memory DataGrid
- Ignite can work with a layer containing HDFS
- Data replication - protection in case of node failure, automatic backupon other nodes

Apache Ignite setup

- download binaries from ignite.apache.org
- start cluster with
  - bin/ignite.sh <optional configuration>
- Maven project
  - Ignite ign = Ignition.start(<optional configuration>);
- default configuration file: config/default-config.xml
- you can launch visor for monitoring:
  - $ bin/ignitevisorcmd.sh

Warning: use Java 8 to avoid problems! (install Java8 and use update-alternatives command to switch between Java versions)

Maven setup

```
<dependency>
    <groupId>org.apache.ignite</groupId>
    <artifactId>ignite-core</artifactId>
    <version>${ignite.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.ignite</groupId>
    <artifactId>ignite-spring</artifactId>
    <version>${ignite.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.ignite</groupId>
    <artifactId>ignite-indexing</artifactId>
    <version>${ignite.version}</version>
</dependency>
```

Ignite

- Ignite is JVM-based. A single JVM represents one or more logical Ignite nodes (most of the time, however, a single JVM runs just one Ignite node)
- Ignite is an elastic, horizontally scalable distributed system that supports adding and removing cluster nodes on demand. Ignite also allows for storing multiple copies of the data, making it resilient to partial cluster failures.
- If the persistence is enabled, then data stored in Ignite will also survive full cluster failures.
- Cluster restarts in Ignite can be very fast, as the data becomes operational instantaneously directly from disk.

Servers and Clients

- Ignite has an optional notion of client and server nodes. Server nodes participate in caching, compute execution, stream processing, etc., while the native client nodes provide the ability to connect to the servers remotely. Ignite native clients support using the whole set of Ignite APIs, including near caching, transactions, compute, streaming, services, etc. from the client side.
- By default, all Ignite nodes are started as server nodes, and client mode needs to be explicitly enabled.

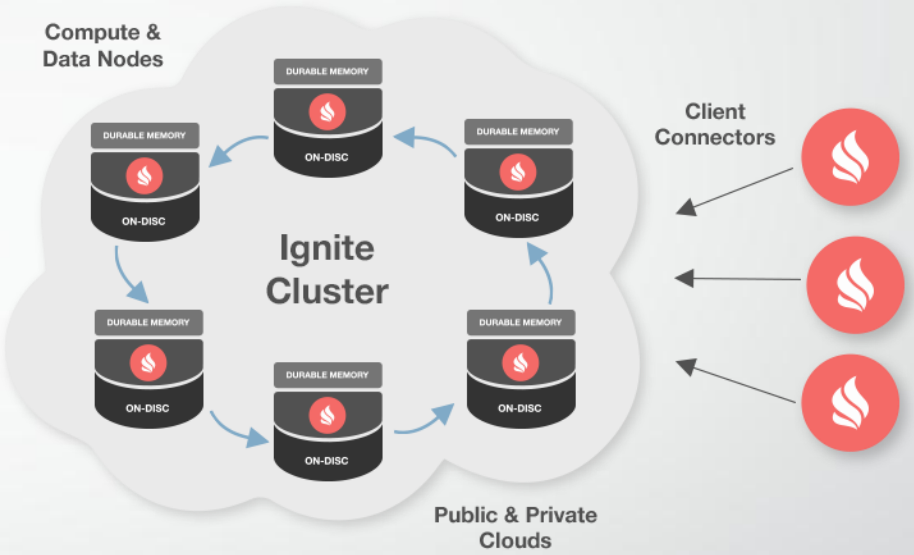
```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
    ...
    <!-- Enable client mode. -->
    <property name="clientMode" value="true"/>
    ...
</bean>
```

```
Ignition.setClientMode(true);

// Start Ignite in client mode.
Ignite ignite = Ignition.start();
```

Cluster discovery

- Multicast Based Discovery
- Static IP Based Discovery
- Multicast and Static IP Based Discovery
- Amazon S3 Based Discovery




```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast
          <property name="multicastGroup" value="228.10.10.157"/>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

Cluster Groups

- In Ignite all nodes are equal by design, so you don't have to start any node in a specific order, or assign any specific roles to them. However, Ignite allows users to logically group cluster nodes for any application-specific purpose.
- You can limit job execution, service deployment, messaging, events, and other tasks to run only within some cluster group. For example, here is how to broadcast a job only to remote nodes (excluding the local node):

```
final Ignite ignite = Ignition.ignite();

IgniteCluster cluster = ignite.cluster();

// Get compute instance which will only execute
// over remote nodes, i.e. not this node.
IgniteCompute compute = ignite.compute(cluster.forRemotes());

// Broadcast to all remote nodes and print the ID of the node
// on which this closure is executing.
compute.broadcast(() -> System.out.println("Hello Node: " + ignite.cluster().localNode().id());
```

First Ignite Compute application

- Example of an application which counts the number of non-white-space characters in a sentence. A sentence is split it into multiple words, every compute job counts the number of characters in each word. In the end, results received from individual jobs are added to get the total count.

```
try (Ignite ignite = Ignition.start("examples/config/example-ignite.xml")) {
  Collection<IgniteCallable<Integer>> calls = new ArrayList<>();

  // Iterate through all the words in the sentence and create Callable jobs.
  for (final String word : "Count characters using callable".split(" "))
    calls.add(word::length);

  // Execute collection of Callables on the grid.
  Collection<Integer> res = ignite.compute().call(calls);

  // Add up all the results.
  int sum = res.stream().mapToInt(Integer::intValue).sum();

    System.out.println("Total number of characters is '" + sum + "'.");
}
```

First Data Grid application

- Example of an application which will put and get values to/from distributed cache, and perform basic transactions.

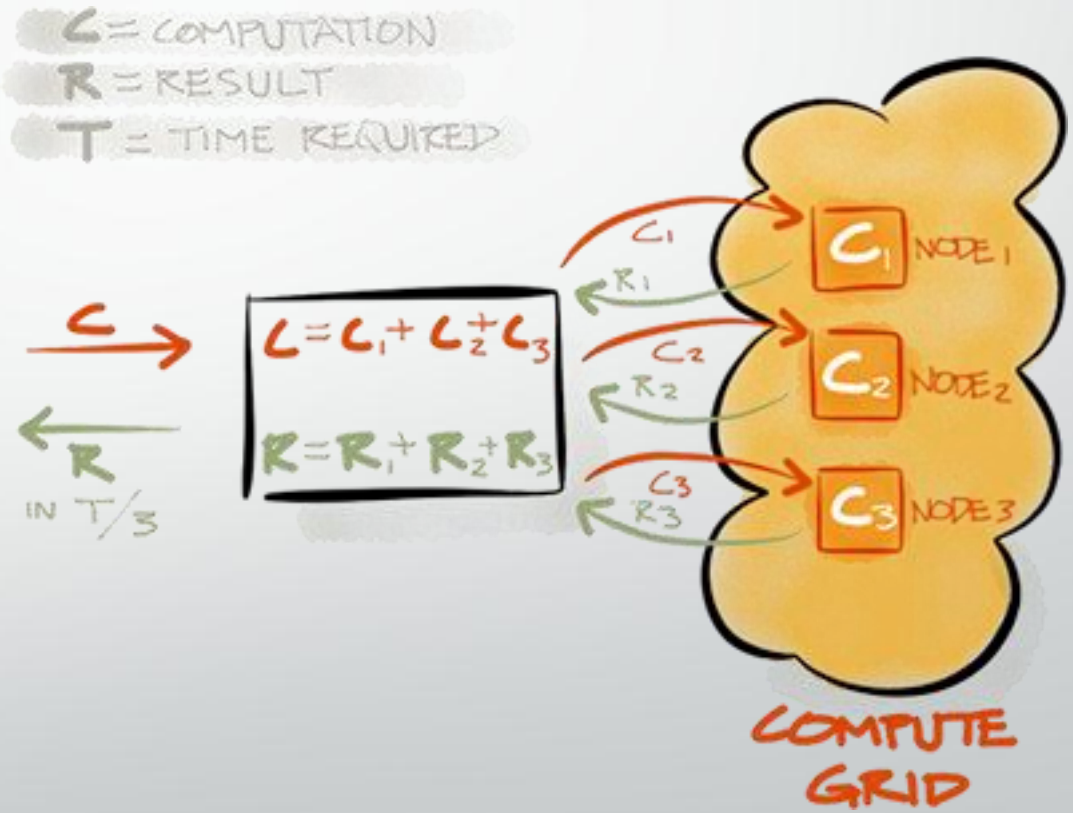
```
try (Ignite ignite = Ignition.start("examples/config/example-ignite.xml")) {
    IgniteCache<Integer, String> cache = ignite.getOrCreateCache("myCacheName");

    // Store keys in cache (values will end up on different cache nodes).
    for (int i = 0; i < 10; i++)
        cache.put(i, Integer.toString(i));

    for (int i = 0; i < 10; i++)
        System.out.println("Got [key=" + i + ", val=" + cache.get(i) + ']');
}
```

Ignite features

- Durable Memory
- Ignite Persistence
- Key-Value Data Grid
- Data Loading & Streaming
- Ignite Compute Grid
- Machine Learning
- Service Grid
- In-Memory File System



Read the documentation!
https://apacheignite.readme.io/v2.3/docs/

Compute Grid & MapReduce

- ComputeTask is the Ignite abstraction for the simplified in-memory MapReduce. Pure MapReduce was never built for performance and only works well when dealing with offline batch oriented processing (e.g. Hadoop MapReduce). With that in mind, Ignite introduced the ComputeTask API, which is a light-weight MapReduce implementation.
- ComputeTask defines jobs to execute on the cluster, and the mappings of those jobs to nodes. It also defines how to process (reduce) the job results. All IgniteCompute.execute(…) methods execute the given task on the grid. User applications should implement the map(…) and reduce(…) methods from the ComputeTask interface.

Ignite examples

- Found in <ignite directory>/examples
- Import to IntelliJ as Maven project
- Build
- Start several Ignite nodes from console (remember to adapt the configuration to the example that you want to run)
- Run the example Java program from IntelliJ, observe the output of the nodes