

Key-value Stores: Practice

Seminar 2 of *NoSQL Databases* (PA195)

David Novak & Vlastislav Dohnal
Faculty of Informatics, Masaryk University, Brno

Agenda

- Key-value stores: properties and representatives
- Riak
 - Mission, types of communication, **HTTP API**
 - Riak **Links** & Riak Search
 - **Java** client
 - Example
 - Internal **features**
- Infinispan
 - Mission, basic **features**
 - Example with **embedded** cache **store**

Technologies Used

- OpenNebula - private **cloud** at FI
 - Infrastructure as a Service
- **SSH** private/public **key** pairs
- **SSH tunneling**
- **curl** to communicate with HTTP REST API

- **Java**
 - IDE

Practical Work with Riak

Riak: Basic Information

- **Developer:** Basho, **open source** community
 - there is a company behind
- **Initial release date:** 2009
 - it is **not** a new (**shaky**) technology
- **License:** Apache 2 + commercial enterprise
 - **for free** but with option to have a paid **support**
- **Language:** Erlang, C, C++, some parts in JavaScript
 - **Efficient**; not possible to embed to e.g. Java application
- **Server OS:** Linux, BSD, Mac OS X, Solaris

Test Environment



Accessible from
outside the FI net

localhost

- Private **cloud** at FI: <https://stratus.fi.muni.cz/>
 - tech info: [link](#)
 - **log in** with your **faculty credentials** (user/password)
- User -> settings -> Public SSH Key
 - upload your public key to the stratus interface
 - take contents of `.ssh/id_rsa.pub` & paste it to WebUI
 - **if you did not find the file**: create a new **ssh key pair**
`nymfe$ ssh-keygen` OR `aisa$ ssh-keygen`
 - by default, stored in `.ssh/id_rsa` and `.ssh/id_rsa.pub`
- Create new VM from template "PA195 CentOS 7"

```
nymfe$ ssh root@172.26.1.ddd  
OR $ ssh -t <LOGIN>@aisa.fi.muni.cz ssh root@172.26.1.ddd
```

ID 648

Basic Riak Commands

stratus

Edit the Riak config: (nodename and internal cluster connection)

```
[root@centos7 ~]# nano /etc/riak/riak.conf
```

change:

```
nodename = <your_name>@<your_IP>
```

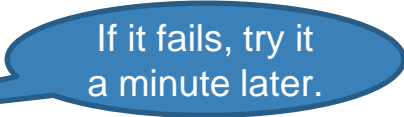
```
listener.protobuf.internal = <your_IP>:8087
```

```
[root@centos7 ~]# riak start
```

```
[root@centos7 ~]# riak ping
```

```
[root@centos7 ~]# riak-admin test
```

```
[root@centos7 ~]# riak stop
```

A blue speech bubble with a tail pointing to the "riak-admin test" command. It contains the text "If it fails, try it a minute later." in white font.

If it fails, try it
a minute later.

documentation: <https://docs.riak.com/riak/kv/2.2.1/>

Admin Web GUI

localhost

- Admin GUI for a **running Riak** instance
 - <http://localhost:8098/admin> or
 - <http://172.26.x.yyy:8098/admin>
- To access the Admin GUI from localhost, use **ssh tunneling**

```
nymfe$ ssh root@172.26.1.ddd -L 8098:localhost:8098
```

OR

```
$ ssh -L 8098:172.26.1.ddd:8098 -t <login>@aisa.fi.muni.cz ssh root@172.26.1.ddd
```


Riak: HTTP API

- Riak **HTTP API** - simple HTTP Restful service
 - HTTP API uses **different HTTP methods**
 - for different **operations**
 - GET (retrieve), PUT (update), POST (create), DELETE (delete)
 - <https://docs.riak.com/riak/kv/2.2.1/developing/api/http/>
 - We will use **curl** (curl --help)
 - a command **line tool** to communicate with server (http(s),...)
- localhost
- **HTTP REST examples** can be accessed on 172.26.1.ddd using
\$ curl http://localhost:8098/...

Riak: Example 1 - Buckets

localhost

- List all the **buckets**

```
curl http://localhost:8098/buckets?buckets=true -s | json_pp
```

- Everyone will use their **own bucket**

```
$ export ARTISTS=${USER}_artists
```

Then, the bucket URL is: `http://localhost:8098/buckets/$ARTISTS`

- Buckets are **created** on the fly
 - Only **when needed**

Riak: Example 2 - Basic Operations

localhost

- **Store** a plain text using a **generated key** (POST)

```
curl -X POST http://localhost:8098/buckets/$ARTISTS/keys/ -d 'text' -i
```

- **List** all **keys** within a bucket

- **not recommended** for production (slow and not reliable)

```
curl http://localhost:8098/buckets/$ARTISTS/keys?keys=true
```

- **Get** an object:

```
curl http://localhost:8098/buckets/$ARTISTS/keys/<key>
```

Riak: Example 3 - Buckets properties

localhost

- Store a **JSON object** under the key *Bruce*

```
curl -X PUT -H "Content-Type: application/json" -d '{"name":  
"Bruce Springsteen", "nationality": "USA"}'  
http://localhost:8098/buckets/$ARTISTS/keys/Bruce
```

- Get **properties** of your bucket:

```
curl http://localhost:8098/buckets/$ARTISTS/props | json_pp
```

- **Change properties** of the bucket:

- Set replication factor to "2"

```
curl -X PUT http://localhost:8098/buckets/$ARTISTS/props -H  
"Content-Type: application/json" -d '{"props": {"n_val": 2}}'
```

Riak: Example 4 - Updates

- **Update** an object = **store** a new value

```
curl -X PUT -H "Content-Type: application/json" -d '{"name":  
"Bruce Springsteen", "nationality": "USA", "date-of-birth":  
"1949-Sep-23"}'  
http://localhost:8098/buckets/$ARTISTS/keys/Bruce
```

- **Check the updated object (with headers):**

```
curl -i http://localhost:8098/buckets/$ARTISTS/keys/Bruce
```

- **Delete** an object:

```
curl -X DELETE  
http://localhost:8098/buckets/$ARTISTS/keys/Bruce
```

Let us Cluster

- Let us **connect** our nodes into one **cluster**
 - <https://docs.riak.com/riak/kv/2.2.1/using/running-a-cluster/>

stratus

```
# riak-admin cluster join xdohnal@172.26.2.115
# riak-admin cluster plan
# riak-admin cluster commit
```

localhost

- Check in Web GUI <http://localhost:8098/admin>

Riak Explorer

- GUI data explorer
 - https://github.com/basho-labs/riak_explorer#installation

stratus

- It is downloaded at the Stratus VM
 - `# cd /root/riak_explorer`
- **Optional** task: Make it run and connect to our cluster
 - Configure it: `# nano etc/riak_explorer.conf`
 - Start it: `# bin/riak_explorer start`

Riak: Links

localhost

- Allow to create **relationships** between objects
 - Like **foreign keys** in RDBMS or **associations** in UML
- Attached to objects via **HTTP header** “Link”
- Add an **album** and link to its **performer**:

```
export ALBUMS=${USER}_albums
```

```
curl -H "Content-Type: application/json" -H "Link:  
</buckets/$ARTISTS/keys/Bruce>; riaktag=\"performer\"" -d  
'{"title": "The River", "year": 1980}'  
http://localhost:8098/buckets/$ALBUMS/keys/TheRiver
```

check:

```
curl -i http://localhost:8098/buckets/$ALBUMS/keys/TheRiver 17
```


Riak: Link Walking

- Locate a **key** and then **continue by link(s)**
 - target specification: /bucket,linktype,[0/1]
- Find the **artist** who **performed** album The River

```
localhost
```

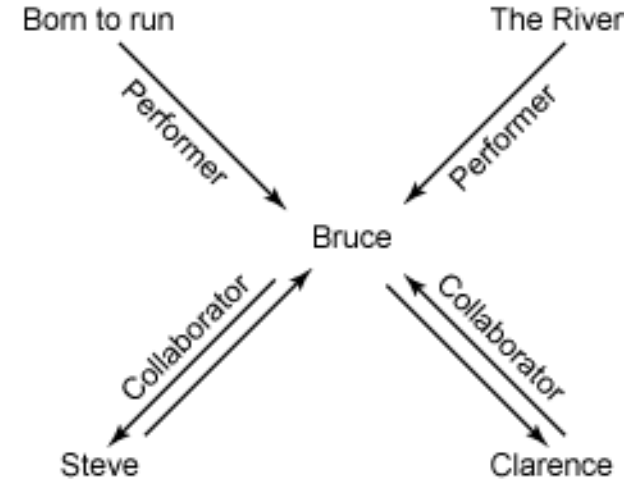
```
curl
```

```
http://localhost:8098/buckets/$ALBUMS/keys/TheRiver/$ARTISTS,performer,1
```

- Restrict to bucket **artists**
- Restrict to tag **performer**
- 1 = include this step to the result (see below)

Riak: Link Walking (2)

- Which artists collaborated with the one who performed album The River
 - We don't have data for this example



localhost

```
curl  
http://localhost:8098/buckets/$ALBUMS/keys/TheRiver/_,performer,0/$ARTISTS,collaborator,1
```

- `_` = wildcard (any bucket)
- `0` = do not include this step to the result

Java Client

- There are **clients** for **many** languages
 - Erlang, Java, Python, Ruby, C, PHP, Node.js, ...
- **Java** Library for communication with Riak
 - Uses **Protocol Buffers** for communication (**port 8087**)
- Download a project from the [study materials](#)
 - uses <https://github.com/basho/riak-java-client>

*If you are not on **nymfe** but on your computer, you must tunnel
!!!*

```
your_computer$ ssh <login>@aisa.fi.muni.cz -L 8087:172.26.1.ddd:8087
```

Java Client (2)

localhost

- Development with IntelliJ IDEA

```
$ unzip riak-client.zip
```

```
$ module add idea-2019.2-loc
```

```
$ idea.sh &
```

Java Client: Test

Use "localhost"
(when forwarding port 8087
to 172.26.1.ddd)



- Basic communication with our Riak instance

```
RiakClient client = RiakClient.newClient("172.26.1.ddd");
Namespace bucket = new Namespace("<your_name>_artists");
Location location = new Location(bucket, "Bruce");
FetchValue fv = new FetchValue.Builder(location).build();
FetchValue.Response response = client.execute(fv);
String obj = response.getValue(String.class);
System.out.println("result: " + obj);

client.shutdown();
```

Task To Do Individually!

localhost

- Create a **database** of all **students** in PA195 (follow Students.java source code):

1. **Download** CSV file with the list of students

https://is.muni.cz/auth/el/fi/podzim2021/PA195/um/seminar-2/seznam_export.csv

2. **Read** the file by **Java**

3. **Store each** record into bucket “<your_login>_students”
 - **Key** is UCO (the second column)

4. **Update** your record and set the name **to your nick**.

5. **Search** the data by a secondary index

- **Stop** your VM in Stratus when you're finished

Practical Work with Infinispan (optional)

Basics

- Developer: **Red Hat, open source** community
 - Originally developed as a memory-based cache for JBoss
- Initial release date: 2009
- License: Apache version 2
- Language: Java
 - **embedding** to **Java** application
 - various **APIs** (REST service, Memcached protocol, Hotrod)

<http://infinispan.org/>

<http://db-engines.com/en/system/Infinispan>

Infinispan: Hello World

- We will use Infinispan as an **embedded** store
- Create a new Java **Maven** project
 - And add dependency package **infinispan-core-8.1.2**
- Or download project from the [study materials](#)
 - Development with IntelliJ IDEA

Infinispan: Hello World

```
public static void main(String args[]){
    Cache<String, Object> store =
        new DefaultCacheManager().getCache();
    store.put("key1", new MyClass("value1"));
    store.put("key2", "value2");
    if (store.containsKey("key1")) {
        Object result = store.get("key2");
        store.removeAsync("key2");
    }
    store.replaceAsync("key2", "value3");
    store.clear();
}
```

Infinispan: Practical Example



- Students example (from Riak)
 - Memory cache
 - Disk-oriented cache
- Use `ispn-config.xml` to configure Infinispan

References

- I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.
- Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 192 p.
- RNDr. Irena Holubova, Ph.D. MMF UK course NDBI040: Big Data Management and NoSQL Databases
- <http://www.slideshare.net/quipo/nosql-databases-why-what-and-when>
- <http://riak.com/>
- <http://infinispan.org/docs/7.0.x/>