Working With Spring Boot and Hazelcast (Distributed Cache)

How to create a simple Spring Boot application, using Hazelcast as a Distributed Cache. This can help improve database performance.

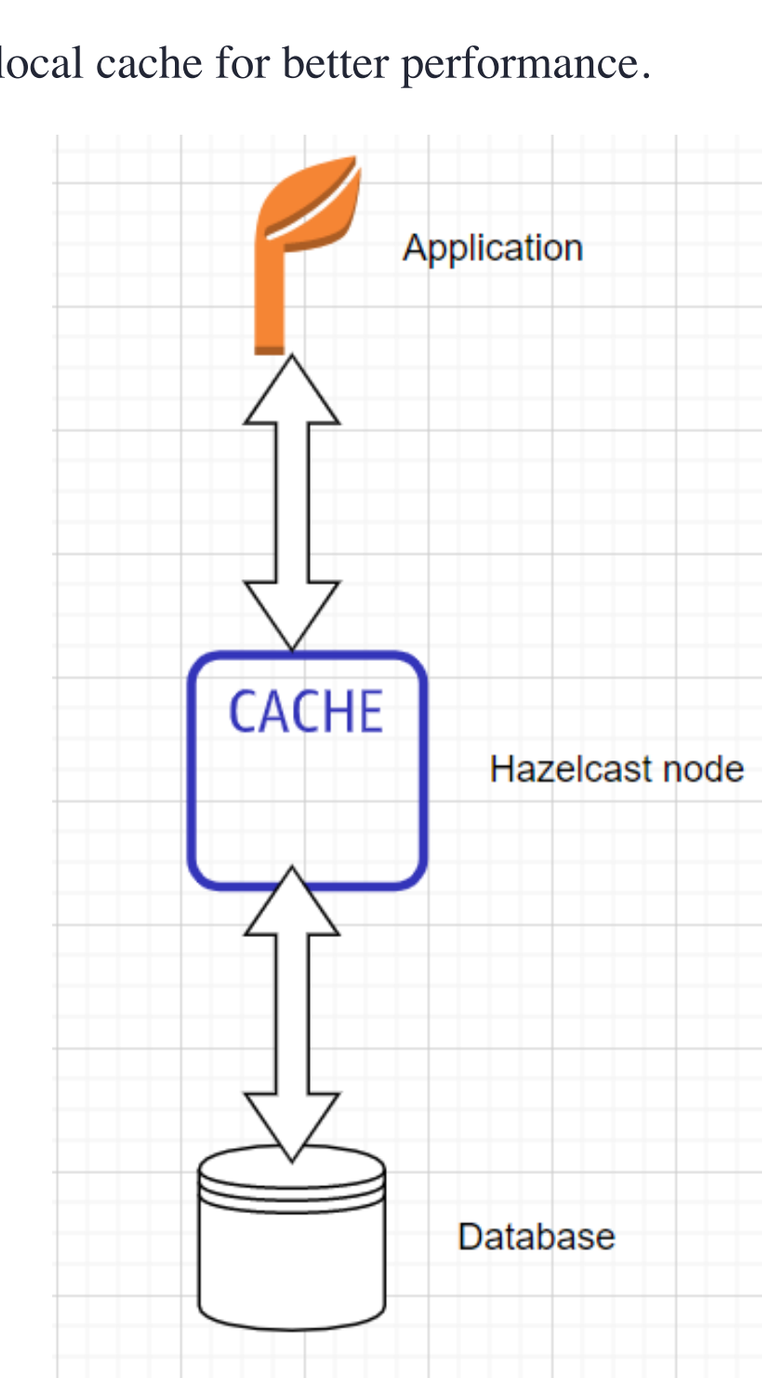
by Diraj Pandey · Feb. 25, 21 · Java Zone · Tutorial

Hazelcast provides central and predictable scaling of applications via in-memory access to **frequently used** data and across an elastically scalable data grid. These techniques both reduce the query load on databases and help improve application speed.

Spring Boot is often used to create stand-alone, production-grade Spring-based Applications that you can 'just run.'

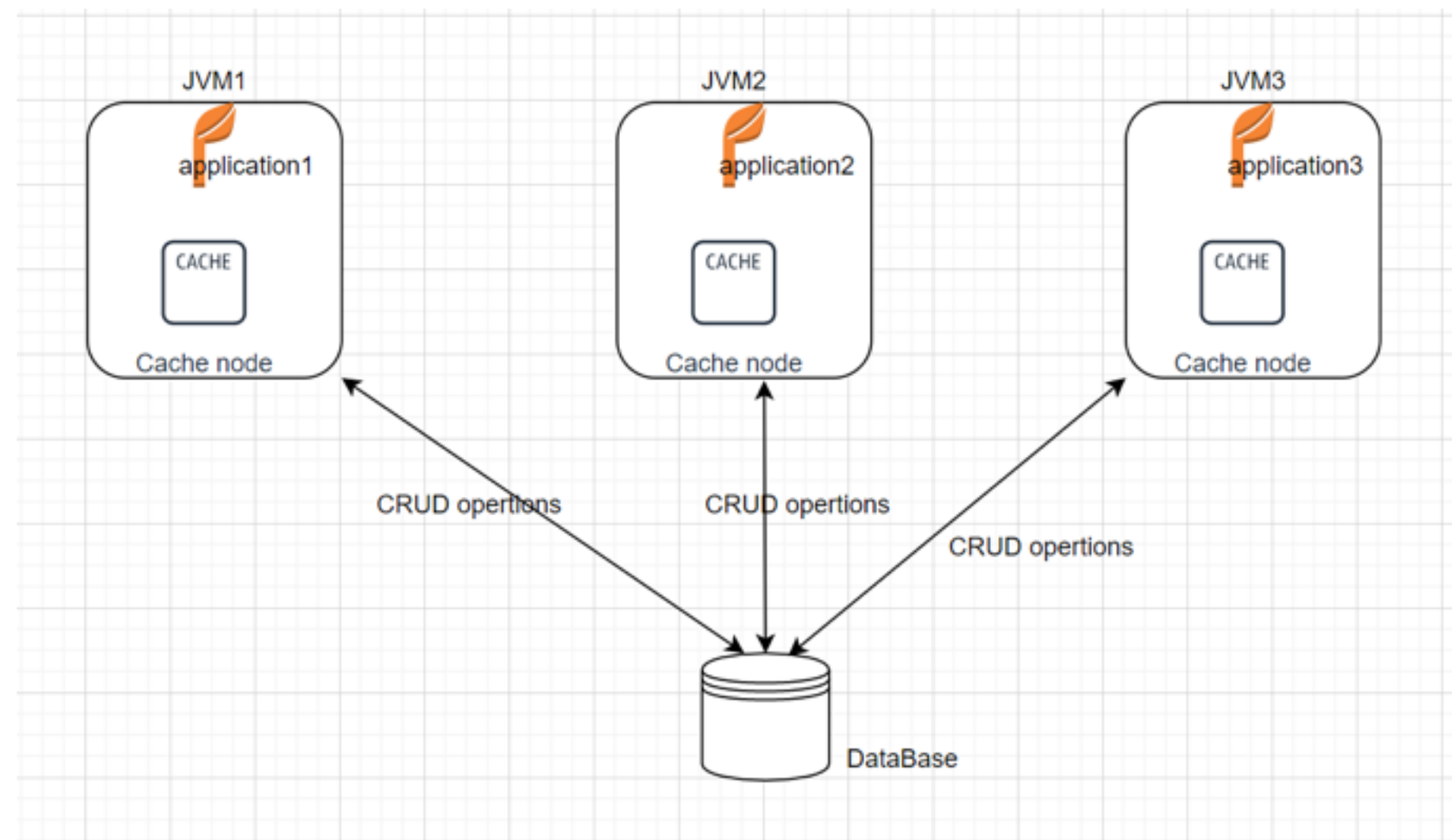
Let us create a simple Spring Boot application with Hazelcast as a Distributed Cache.

Below is an implementation of a local cache for better performance.



Here, when a request is made from an application, if that data is available in the cache, then we don't have to make a call to the database, or if data is not available in the cache, then a request is made to the database.

We deploy the application as a distributed system, as below, for better load balancing.



Implementation

Here is the implementation of a simple spring boot application.

First, we make changes in the pom file by adding a Hazelcast dependency, as below:

```xml
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast</artifactId>
</dependency>
```

We also add a plugin where the entry point to the application is provided in the distributed system. Since `DoctorApplication` is the main class it is added in the plugin as below:

```xml
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <mainClass>com.rbitcs.DoctorApplication</mainClass>
    </configuration>
</plugin>
```

POM file should look like below:

pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.rbitcs</groupId>
    <artifactId>hazelcast-springboot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.5.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>com.hazelcast</groupId>
            <artifactId>hazelcast</artifactId>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <mainClass>com.RBITCS.DoctorApplication</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

We will be using IMAP provided by hazelmap as the cache. IMAP implements a map interface. Below is the Spring Boot application where beans use IMAP for caching in configuring the Hazelcast instance.

```java
package com.rbitcs;

import java.util.HashMap;
import java.util.Map;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.ManagementCenterConfig;
import com.hazelcast.core.Hazelcast;
import com.hazelcast.core.HazelcastInstance;
import com.hazelcast.core.IMap;
import com.codeusingjava.model.UserAccount;

import com.rbitcs.model.Doctors;

@SpringBootApplication
public class HospitalApplication {

    public static void main(String[] args) {
        SpringApplication.run(HospitalApplication.class, args);
    }

    @Bean
    public Config hazelCastConfig() {
        return new Config().setManagementCenterConfig(
                new ManagementCenterConfig().setEnabled(true).setUrl("http://localhost:8080/hazelcast-
mancenter"));

    }

    @Bean
    public HazelcastInstance hazelcastInstance(Config hazelCastConfig) {
        return Hazelcast.newHazelcastInstance(hazelCastConfig);
    }

    @Bean
    public Map<String, Doctor> doctorMap(HazelcastInstance hazelcastInstance) {
        return hazelcastInstance.getMap("doctorMap");
    }
}
```

Since we are making use of IMAP we need to serialize the object. Here, Doctor object should be serializable.

```java
package com.rbitcs.model;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "doctor")
public class Doctor implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "doctorNumber")
    private String doctorNumber;

    @Column(name = "name")
    private String name;

    @Column(name = "address")
    private String address;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public long getBalance() {
        return balance;
    }

    public void setBalance(long balance) {
        this.balance = balance;
    }

    public String getDoctorNumber() {
        return doctorNumber;
    }

    public void setDoctorNumber(String doctorNumber) {
        this.doctorNumber = doctorNumber;
    }
}
```

Let us create a controller and auto-wire `doctorMap`:

```java
package com.rbitcs.controller;

import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.rbitcs.db.DoctorRepository;
import com.rbitcs.model.Doctor;

@RestController
@RequestMapping(path = "doctors")
public class DoctorController {

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private Map<String, Doctor> doctorMap;

    @GetMapping(path = { "/get/{doctorNumber}" })
    public Doctor getDoctor(@PathVariable("doctorNumber") String doctorNumber) {
        /First call is to check if doctormap has doctor details if yes, return the value otherwise call
database.
        Doctor doctor = (doctortMap.get(doctorNumber) != null) ? doctorMap.get(doctorNumber)
                : DoctorRepository.findByDoctorNumber(doctorNumber);
        return doctor;
    }

    @PostMapping("/add")
    public void createDoctor(@RequestBody Doctor doctor) {
        //save doctor details in cache
        doctorMap.put(doctor.getDoctorNumber(), doctor);
        doctorRepository.save(doctor);
    }

    @DeleteMapping(path = { "/delete/{doctorNumber}" })
    public Doctor deleteDoctor(@PathVariable("doctorNumber") String doctorNumber) {
        //remove doctor details from both cache and database
        doctorMap.remove(doctorNumber);
        return doctorRepository.deleteByDoctorNumber(doctorNumber);
    }
}
```

Start Application

Nice! We have made all the necessary changes.

Let us start the application with the below commands:

```
java -Dserver.port='8081' -jar boot-hazel-0.0.1-SNAPSHOT.jar
```

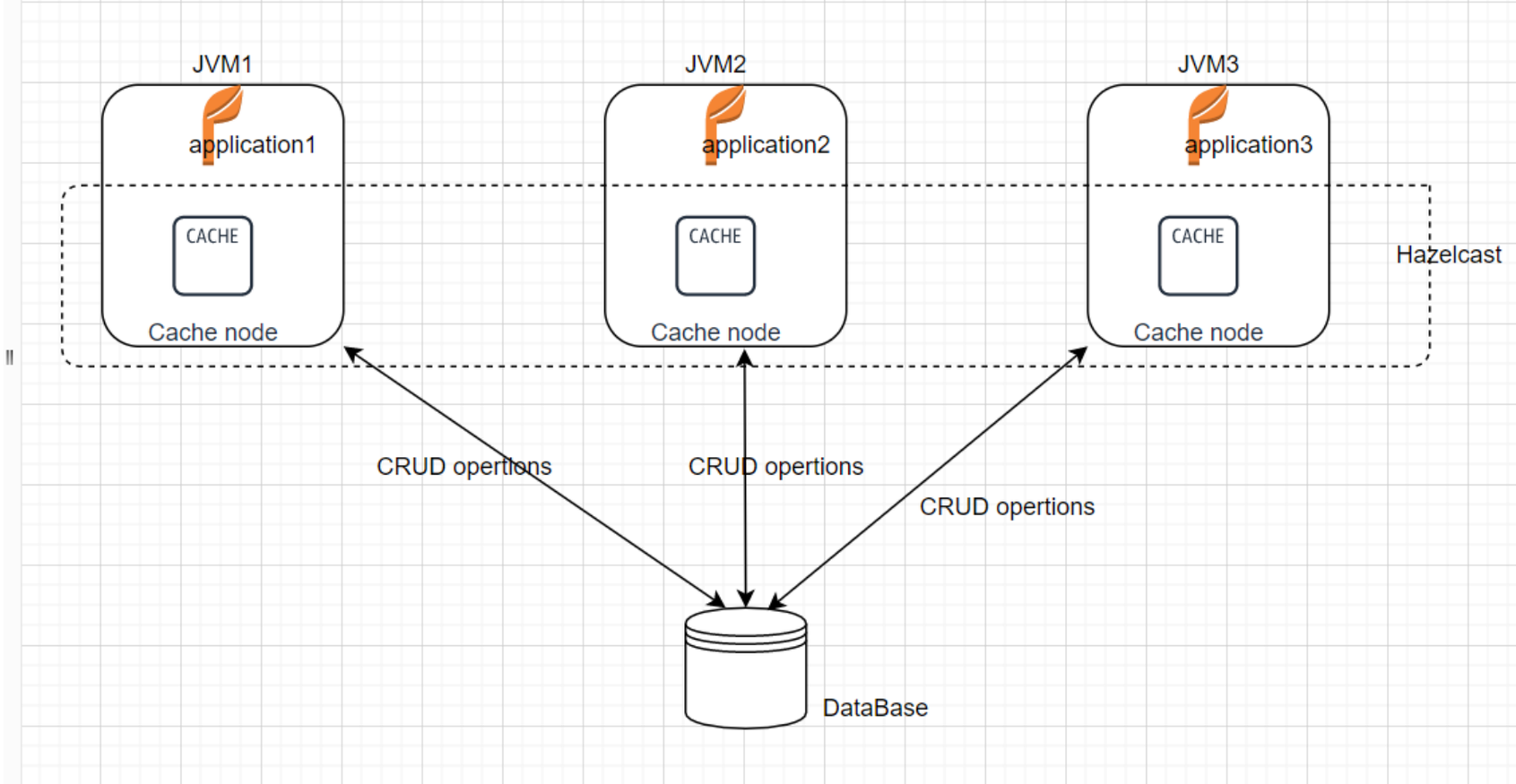
```
java -Dserver.port='8082' -jar boot-hazel-0.0.1-SNAPSHOT.jar
```

```
java -Dserver.port='8083' -jar boot-hazel-0.0.1-SNAPSHOT.jar
```

Let's say we have started 3 instances at 3 different ports (8081, 8082, 8083).

When a doctor's details are stored via the 8081 port application, then the same object can be retrieved from the 8083 port or 8082 port application from the cache and it does not need to make a call to the database.

After the above implementation, the architecture looks like below:



Here all the nodes are synchronized and the same date will be retrieved when requested from any of the JVM with better performance and better load balance.

Topics: HAZELCAST, TUTORIAL, SPRING BOOT, SPRING BOOT APPLICATION