Tanzu GemFire

Building Faster Cloud Native Applications at Scale with VMware Tanzu GemFire

Getting Started

Download Source Code

https://github.com/ggreen/spring-geode-showcase.git

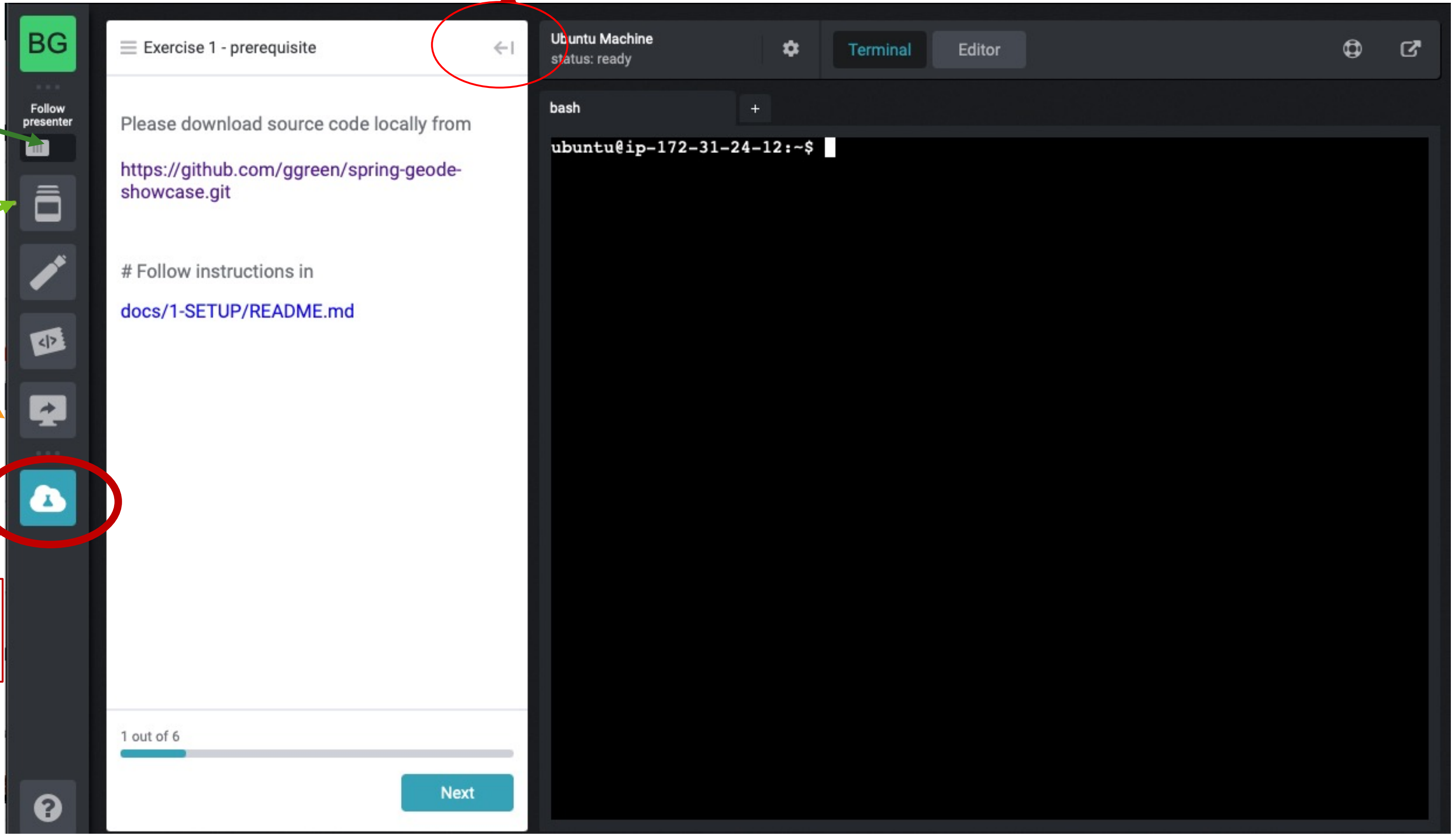
Click arrow If you do not see the exercises

Following presenter

See slides

See presenter screen

Click on My lab



Please download source code locally from

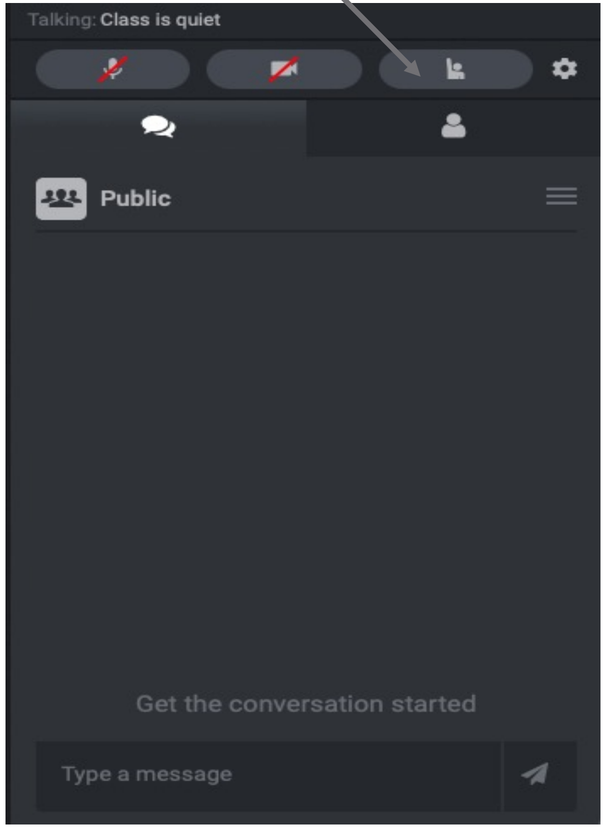
https://github.com/ggreen/spring-geode-showcase.git

Follow instructions in

docs/1-SETUP/README.md

```
ubuntu@ip-172-31-24-12:~$
```

If you need assistance



VMware Tanzu – Data Services

VMware Tanzu

Infrastructure for running modern apps and backing services with consistent, conformant Kubernetes everywhere.

Data Management
Management for Tanzu Data Services instances

GemFire
Fast In-Memory data store for Caching, Transactional and NoSQL support powered by Apache Geode

I need a fast data store

SQL
Relational MySQL or Postgres database for Transactional or Analytic data processing

I need to replatform a relational database

Greenplum
Massively Parallel Processing (MPP) Postgres for Big Data store for analytics, Machine Learning and Artificial Intelligence

I need to drive analytic value of out tons of existing data

Rabbit MQ
High throughput broker for reliable messaging delivery

I need reliable messaging delivery

Spring Cloud Data Flow
Data integration orchestration service for dynamically building data pipelines

I need flexible and manageable data integrations

Features

- ✓ Cloud deployed backing-services
- ✓ On-Premise and Multi-Cloud
- ✓ Based on open source
- ✓ Scaling
- ✓ HA - Fault Tolerant
- ✓ Secured access
- ✓ World Class Support

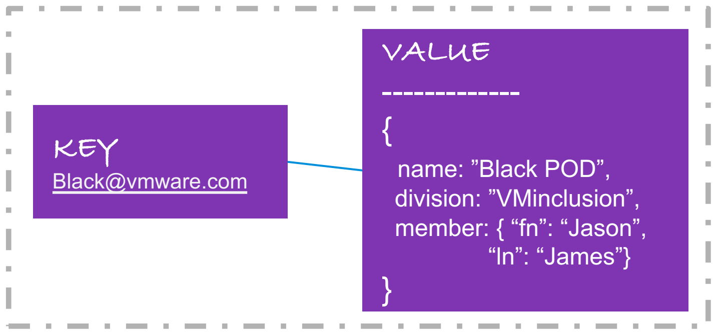
Tanzu GemFire

Use Cases

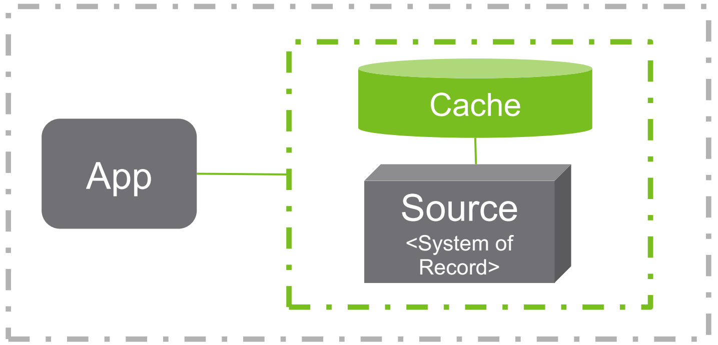
I need a fast data store?

- **NO SQL** data store
  - Fast lookup by key identifiers In-Memory
  - SQL like query (Object Query Language - OQL)
  - Full text-search access
  - Horizontal scalability support
  - High-Availability & Fault Tolerance support
  - WAN replication
  - Triggers/Event notations
  - Stored procedure data processing need
- **Cache** data store
  - API exposed to user interfaces with a real-time interface
  - < 1 second response times
  - Expire cached entries as needed
- Transactional **Operational** data store
  - Persistent with STRONG Consistency – ACID compliance
  - System of record

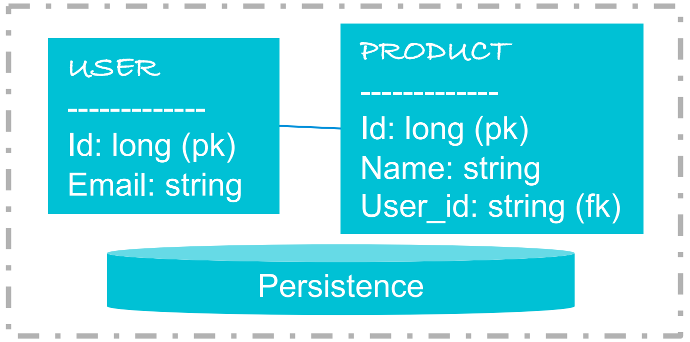
NoSQL Data Store



Cache Data Store



Operational Data Store



GemFire

Fundamentals

Core components

- Data Node – Cache Server – In-memory data storage
- Locator – clients and data nodes controller

Add Data Nodes as needed

- Handle data growth
- Increased processing demands of clients
- Supports resiliency

GemFire cluster

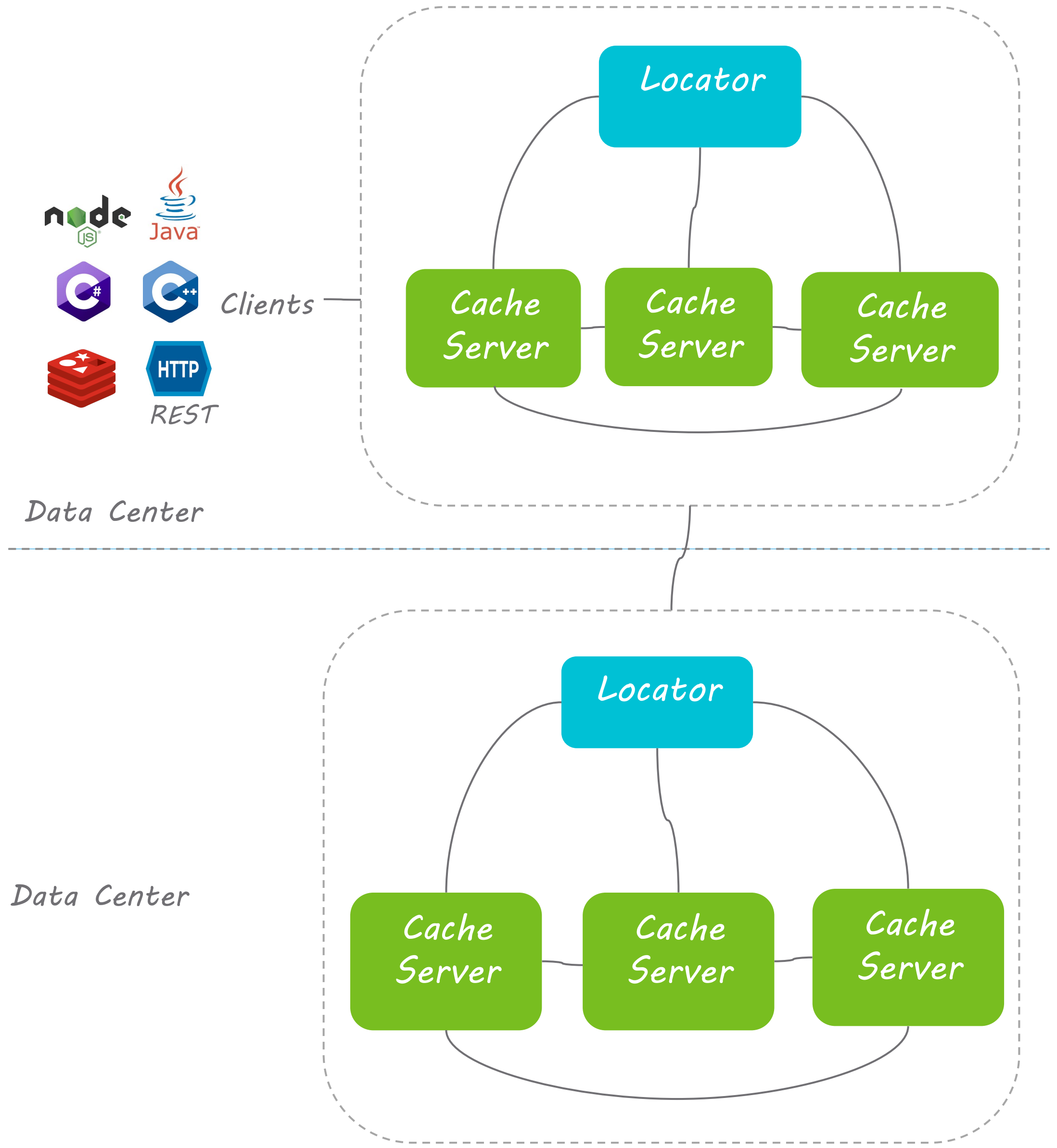
- Connected locators and data nodes

Clients

- Various supported client libraries

WAN Replication

- Replication data across data centers for disaster recovery (DR)
- Active-Active or Active-Passive



Regions

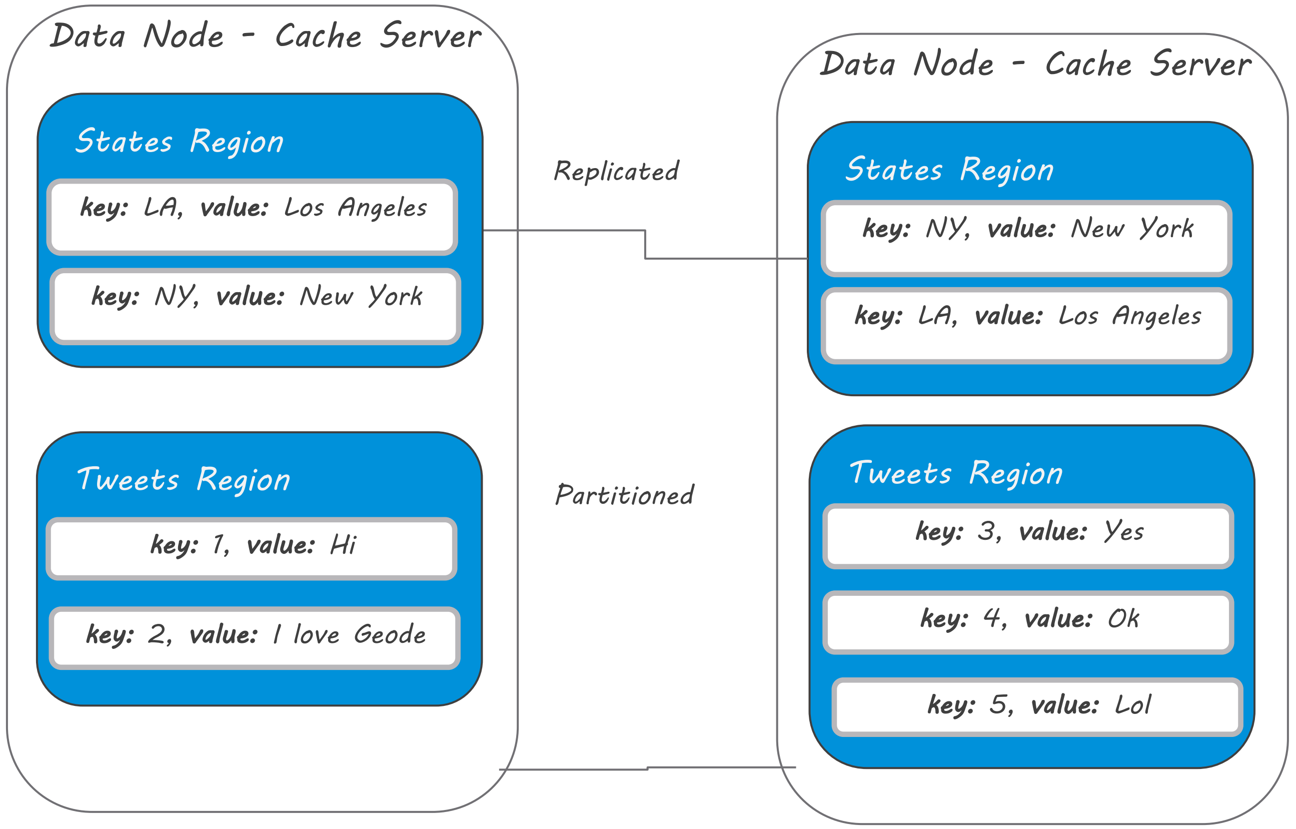
GemFire Region is a database table like data store represented in key/value java.util.Map structure

- Region supports querying
  - Ex: select * from /states where code in (‘NY’, ‘LA’)
  - Ex: lucene search --regionName=/tweet -queryStrings=”*Spring*" --defaultField=tweet
- Events
  - Listeners – triggers data events to client or server-side code
  - Continuous Query – client-side code alerting based on select statements
    - Ex: select * from /tweet where …
- Transaction support

```
//Java Code
Region<String,State> region;

region.put(state.code, state);
```

Cluster Data Policies

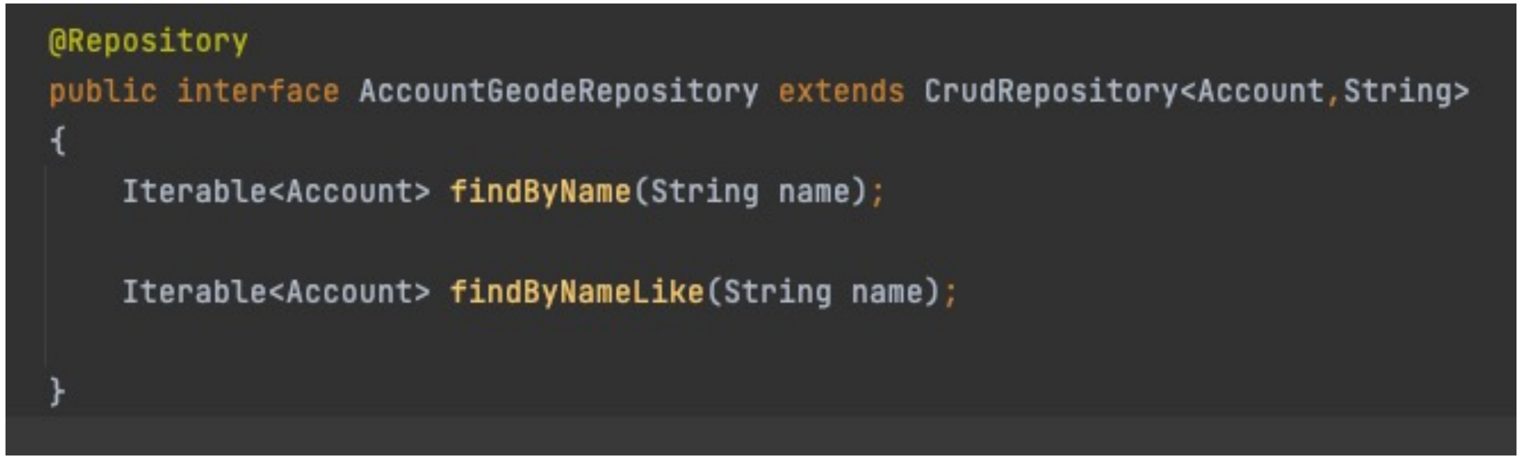
- Replicated Region
  - Full copy on each JVM peer
- Partitioned Region
  - Each peer only stores parts of the region contents



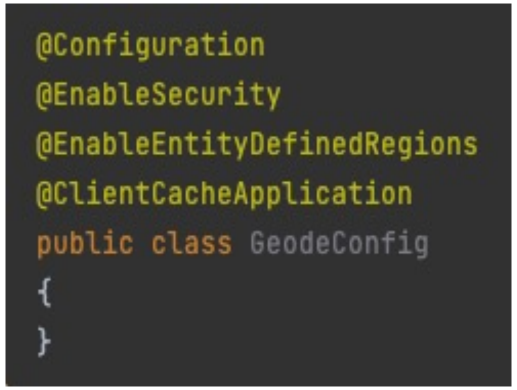
Spring Data Geode

Spring based abstraction layer

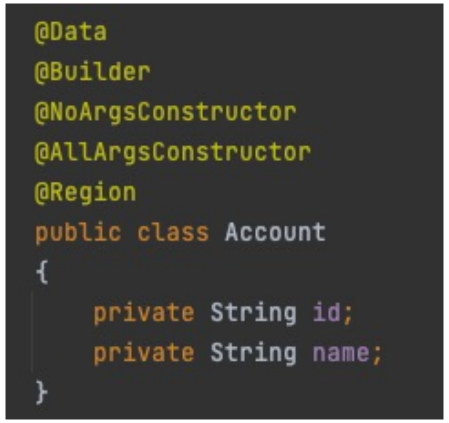
- Bootstrapping GemFire
- Spring Data template-based CRUD POJO access, exception translation, transaction management, and query operations.
- Incorporate best practice serialization of managed objects.
- Event driven abstraction using Continuous Query (CQ) to process a stream of events based on interest defined thru the OQL (Object Query Language).



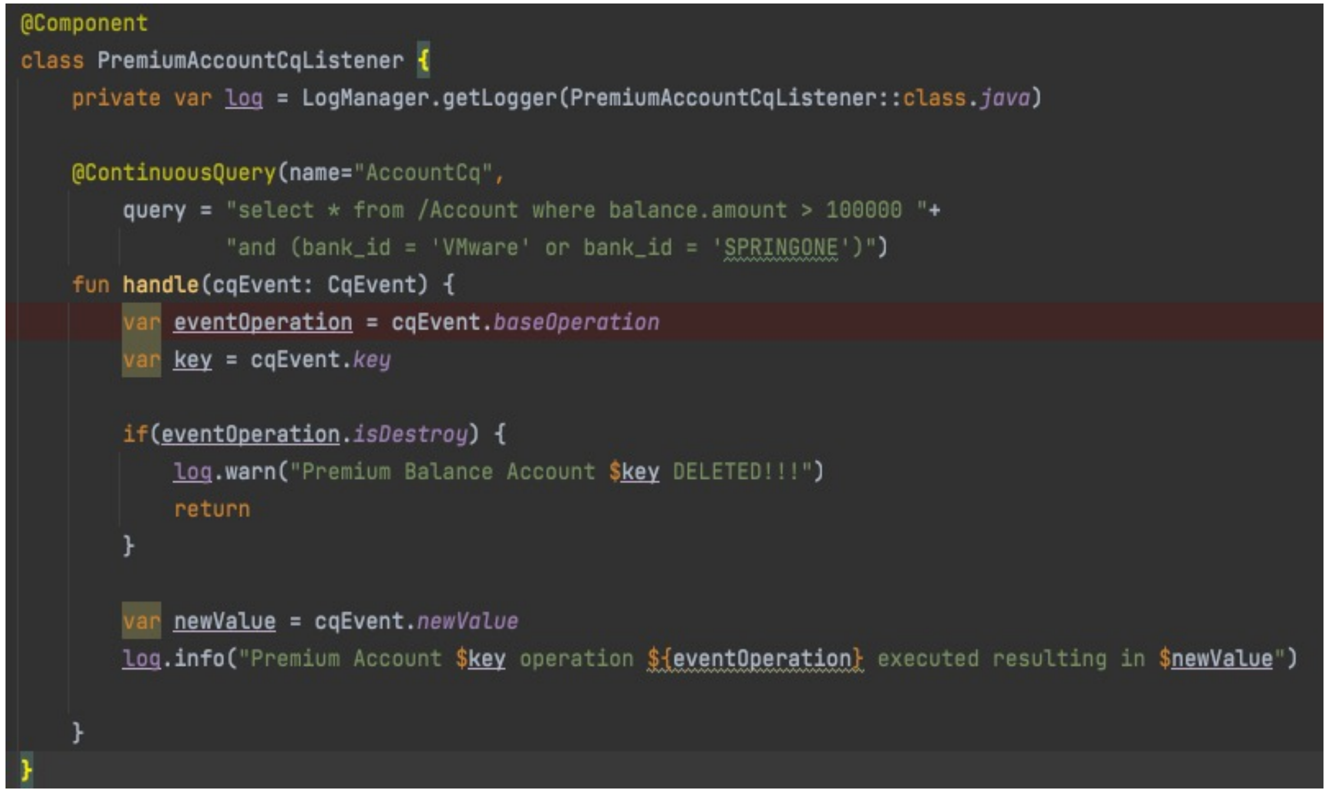
```
@Repository
public interface AccountGeodeRepository extends CrudRepository<Account,String>
{
    Iterable<Account> findByName(String name);

    Iterable<Account> findByNameLike(String name);

}
```



```
@Configuration
@EnableSecurity
@EnableEntityDefinedRegions
@ClientCacheApplication
public class GeodeConfig
{
}
```



```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Region
public class Account
{
    private String id;
    private String name;
}
```

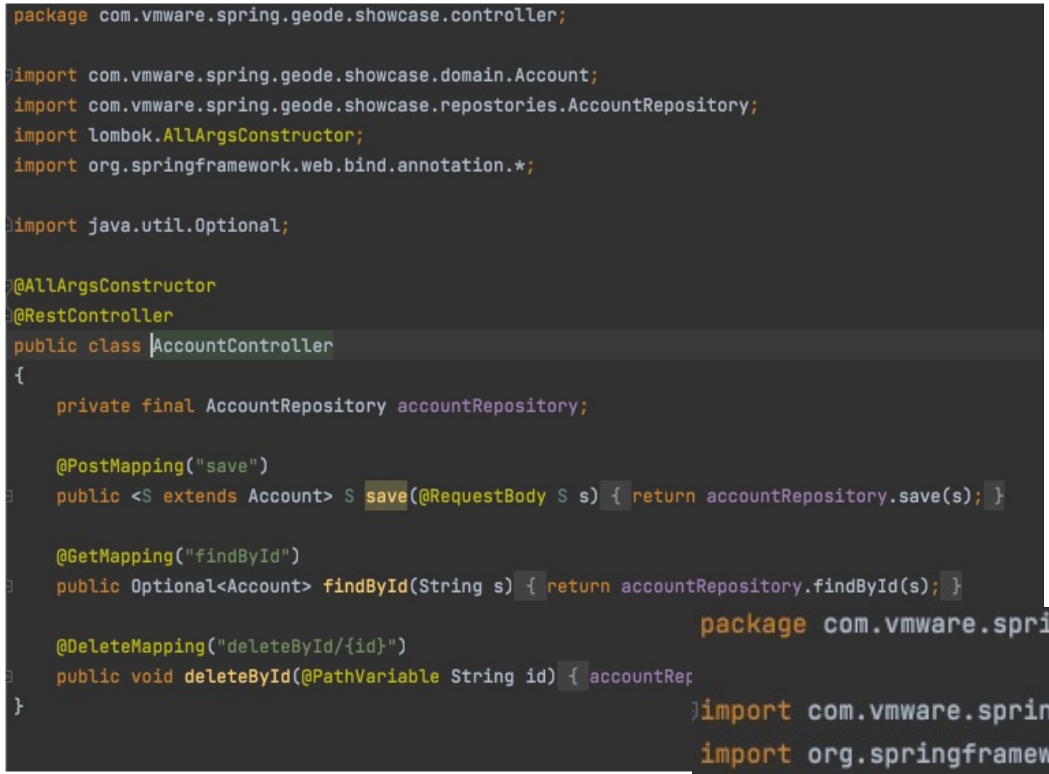


```
@Component
class PremiumAccountCqListener {
    private var log = LogManager.getLogger(PremiumAccountCqListener::class.java)

    @ContinuousQuery(name="AccountCq",
        query = "select * from /Account where balance.amount > 100000 "+
                "and (bank_id = 'VMware' or bank_id = 'SPRINGONE')")
    fun handle(cqEvent: CqEvent) {
        var eventOperation = cqEvent.baseOperation
        var key = cqEvent.key

        if(eventOperation.isDestroy) {
            log.warn("Premium Balance Account $key DELETED!!!")
            return
        }

        var newValue = cqEvent.newValue
        log.info("Premium Account $key operation ${eventOperation} executed resulting in $newValue")

    }
}
```

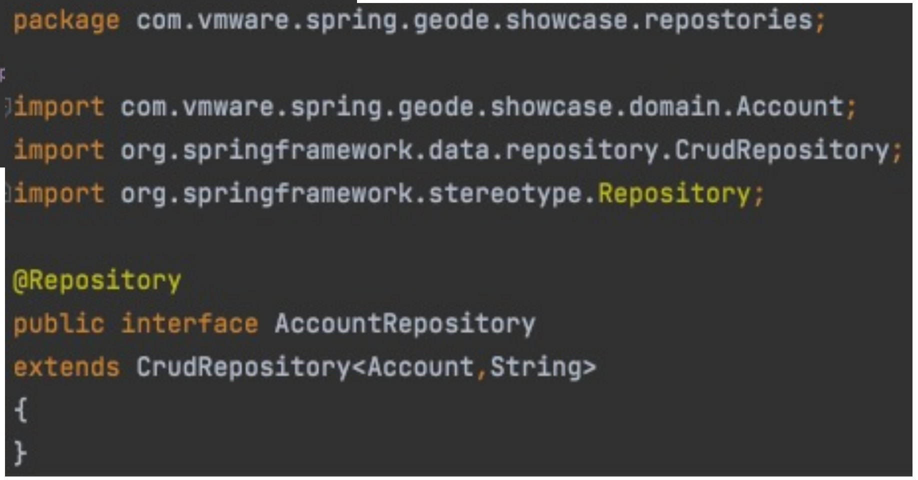
https://github.com/ggreen/spring-geode-showcase.git

Exercise

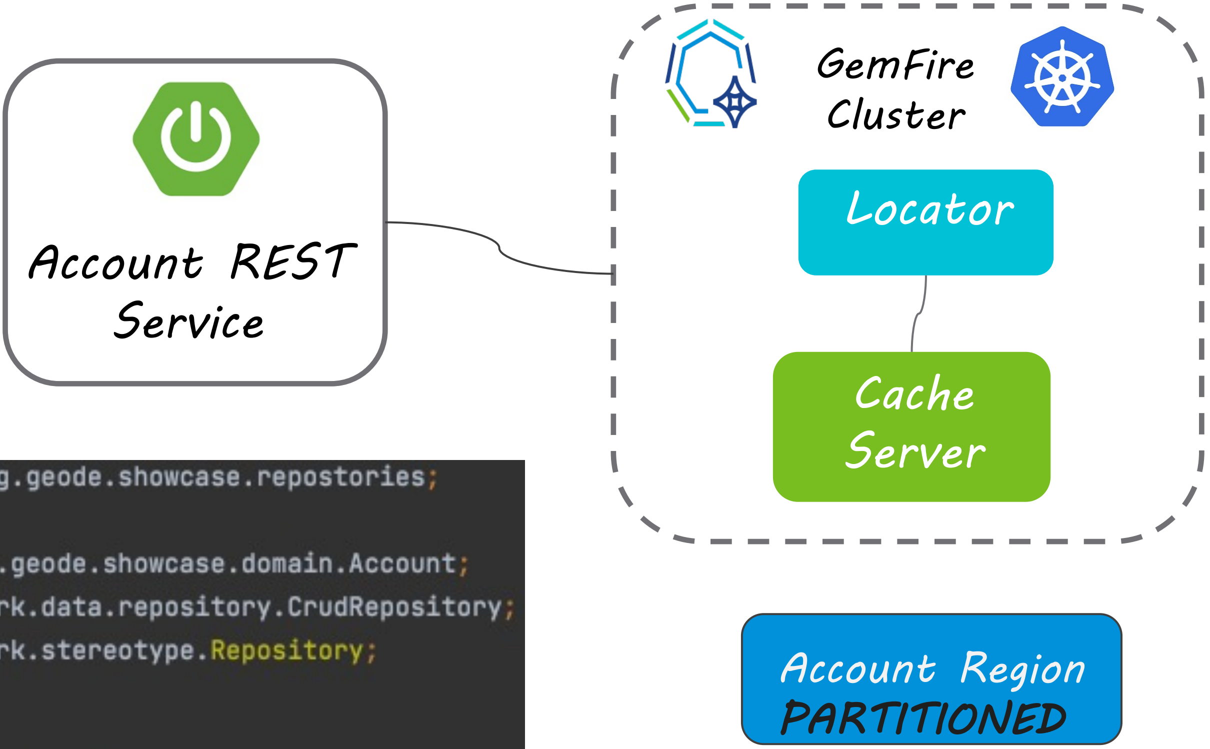
GemFire Cluster - Setup



```
package com.vmware.spring.geode.showcase.controller;

import com.vmware.spring.geode.showcase.domain.Account;
import com.vmware.spring.geode.showcase.repostories.AccountRepository;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@AllArgsConstructor
@RestController
public class AccountController
{
    private final AccountRepository accountRepository;

    @PostMapping("save")
    public <S extends Account> S save(@RequestBody S s) { return accountRepository.save(s); }

    @GetMapping("findById")
    public Optional<Account> findById(String s) { return accountRepository.findById(s); }

    @DeleteMapping("deleteById/{id}")
    public void deleteById(@PathVariable String id) { accountRep
}
```



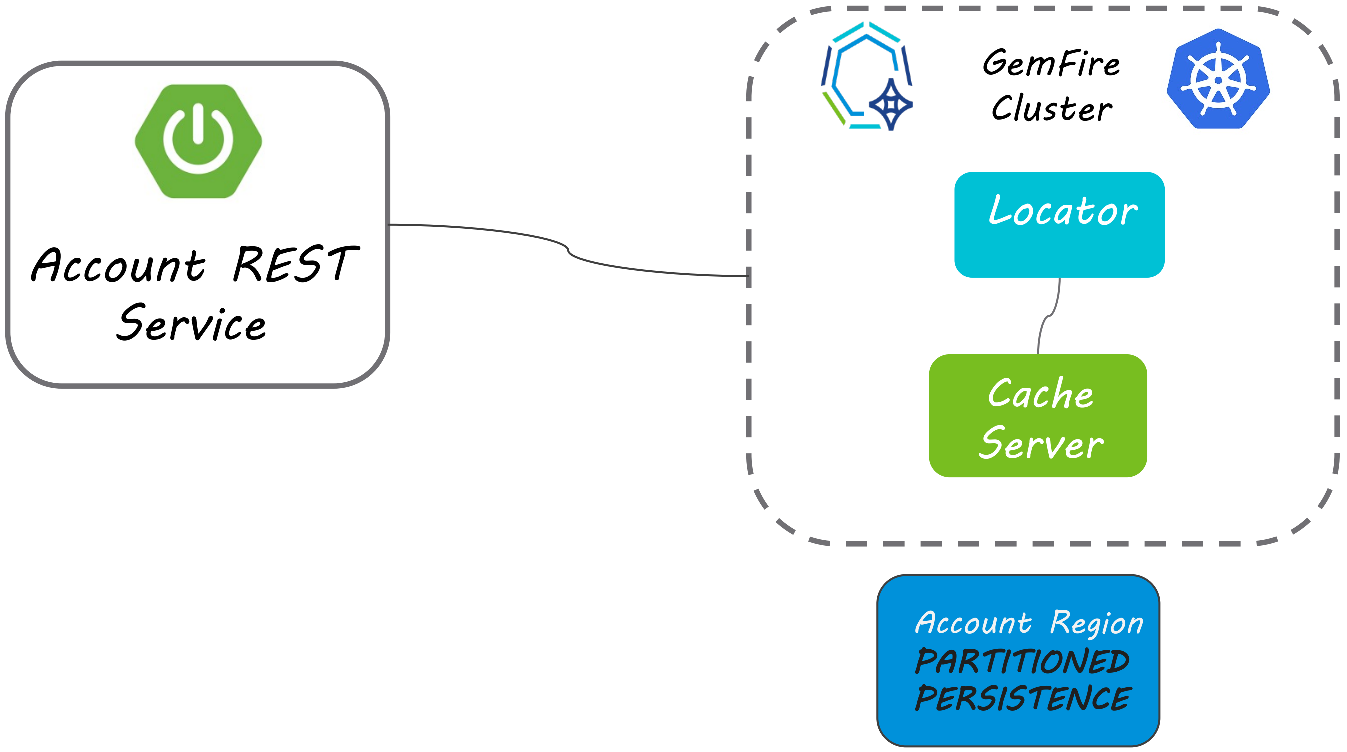
```
package com.vmware.spring.geode.showcase.repostories;

import com.vmware.spring.geode.showcase.domain.Account;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface AccountRepository
extends CrudRepository<Account,String>
{
}
```



https://github.com/ggreen/spring-geode-showcase.git

Exercise

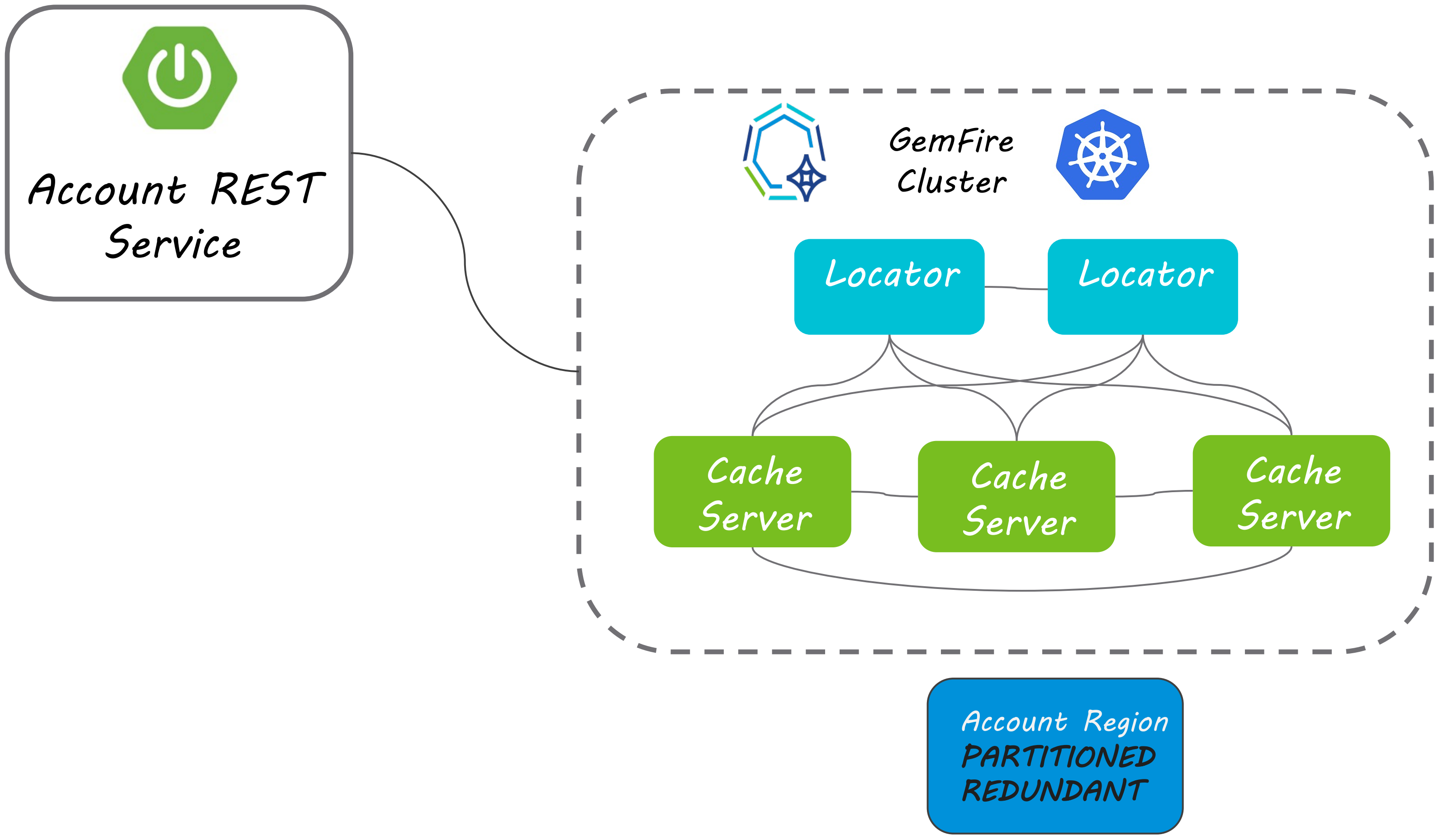
Persistence



https://github.com/ggreen/spring-geode-showcase.git

Exercise

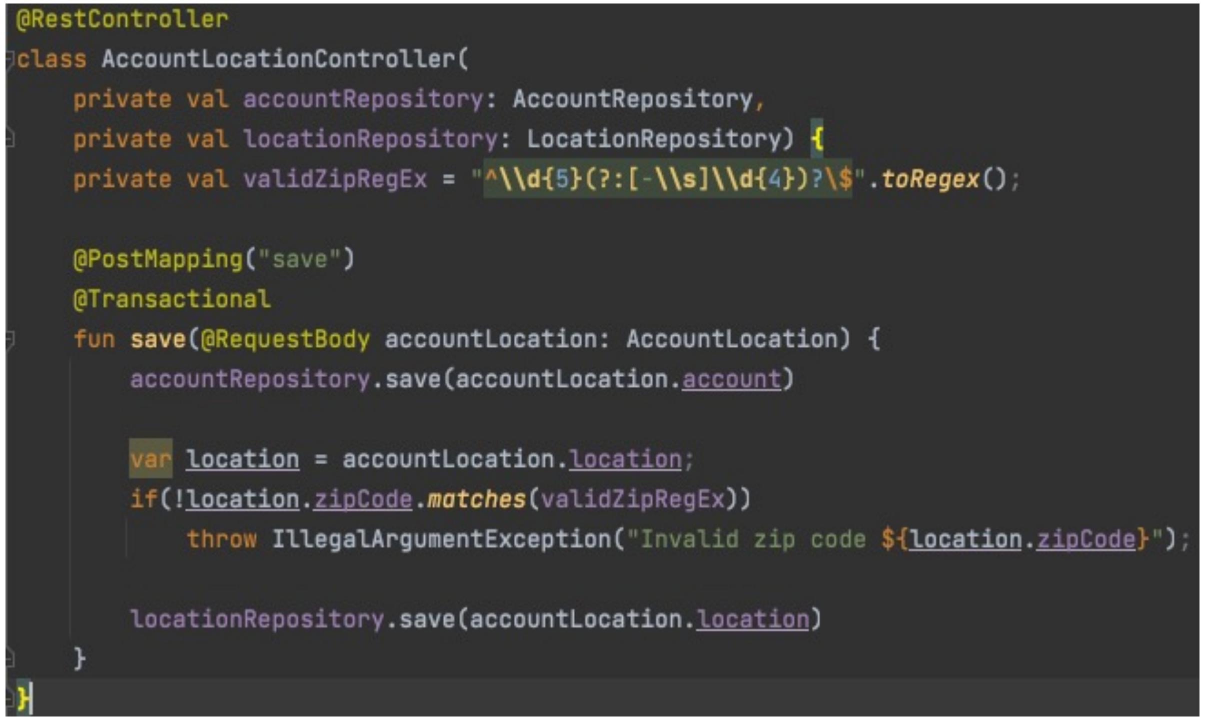
Scalability/High Availability



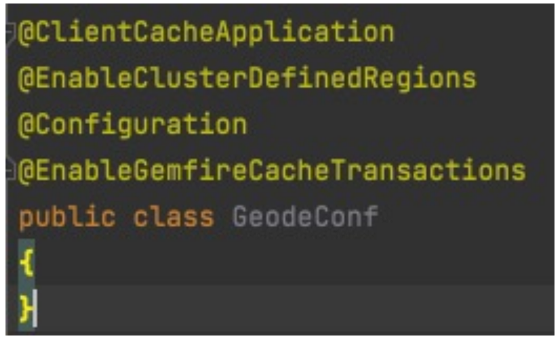
https://github.com/ggreen/spring-geode-showcase.git

Exercise

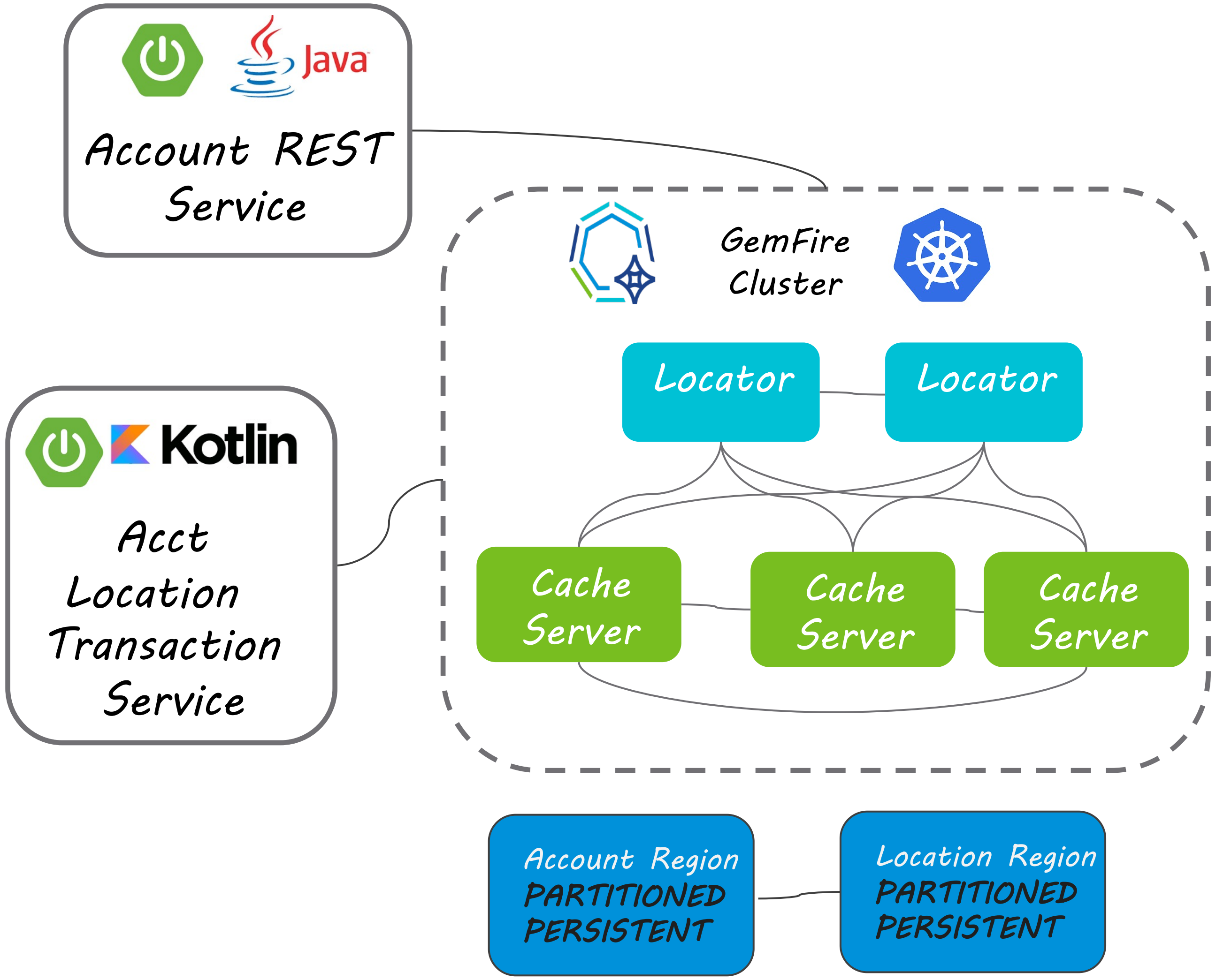
Transaction



```kotlin
@RestController
class AccountLocationController(
    private val accountRepository: AccountRepository,
    private val locationRepository: LocationRepository) {
    private val validZipRegEx = "^\\d{5}(?:[-\\s]\\d{4})?\$".toRegex();

    @PostMapping("save")
    @Transactional
    fun save(@RequestBody accountLocation: AccountLocation) {
        accountRepository.save(accountLocation.account)

        var location = accountLocation.location;
        if(!location.zipCode.matches(validZipRegEx))
            throw IllegalArgumentException("Invalid zip code ${location.zipCode}");

        locationRepository.save(accountLocation.location)
    }
}
```



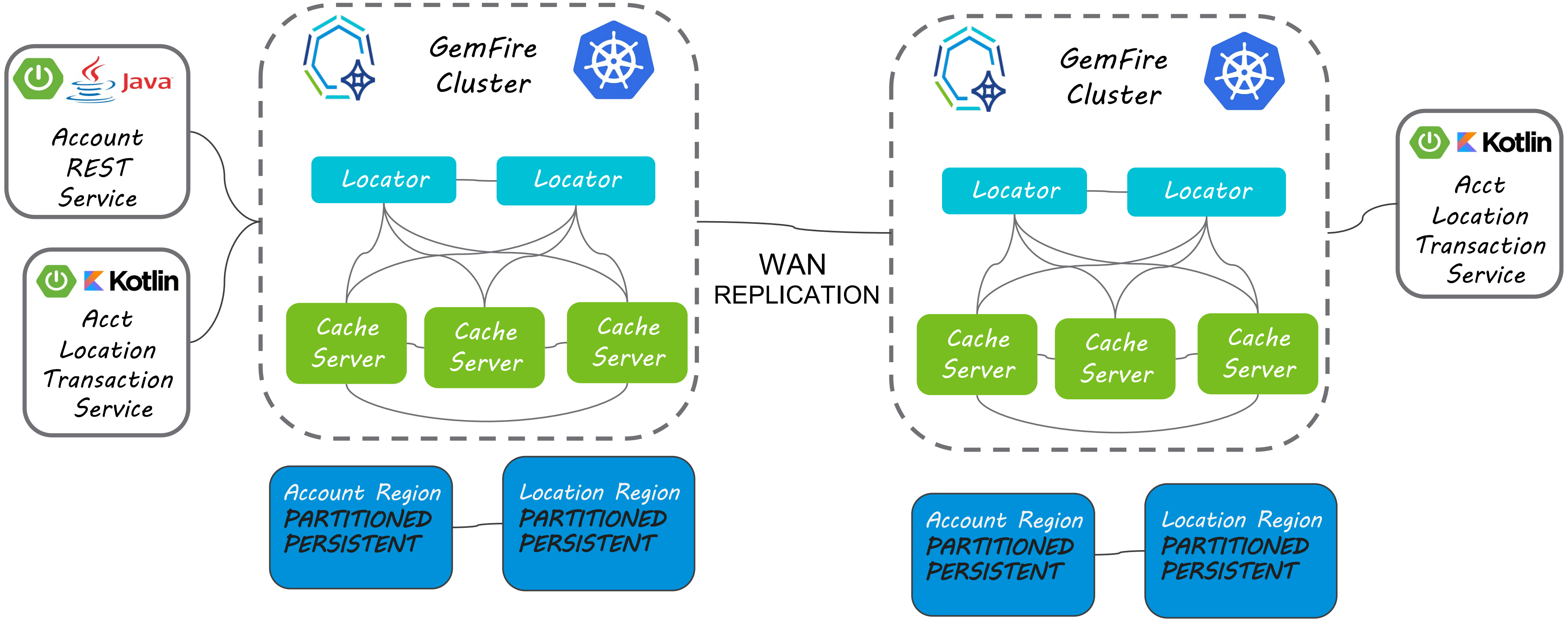
```java
@ClientCacheApplication
@EnableClusterDefinedRegions
@Configuration
@EnableGemfireCacheTransactions
public class GeodeConf
{
}
```


Java
Account REST Service
GemFire Cluster
Locator
Locator
Kotlin
Acct Location Transaction Service
Cache Server
Cache Server
Cache Server
Account Region PARTITIONED PERSISTENT
Location Region PARTITIONED PERSISTENT

https://github.com/ggreen/spring-geode-showcase.git

Exercise

WAN Replication

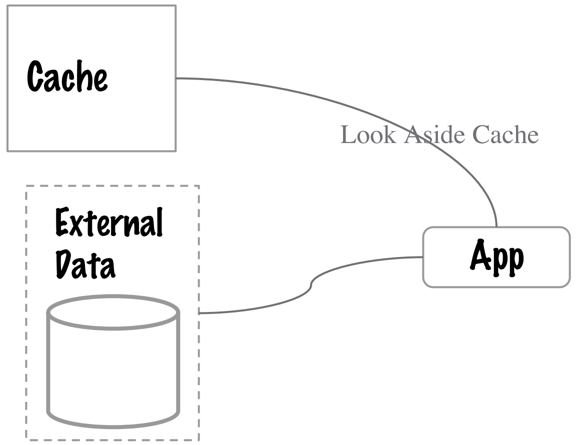


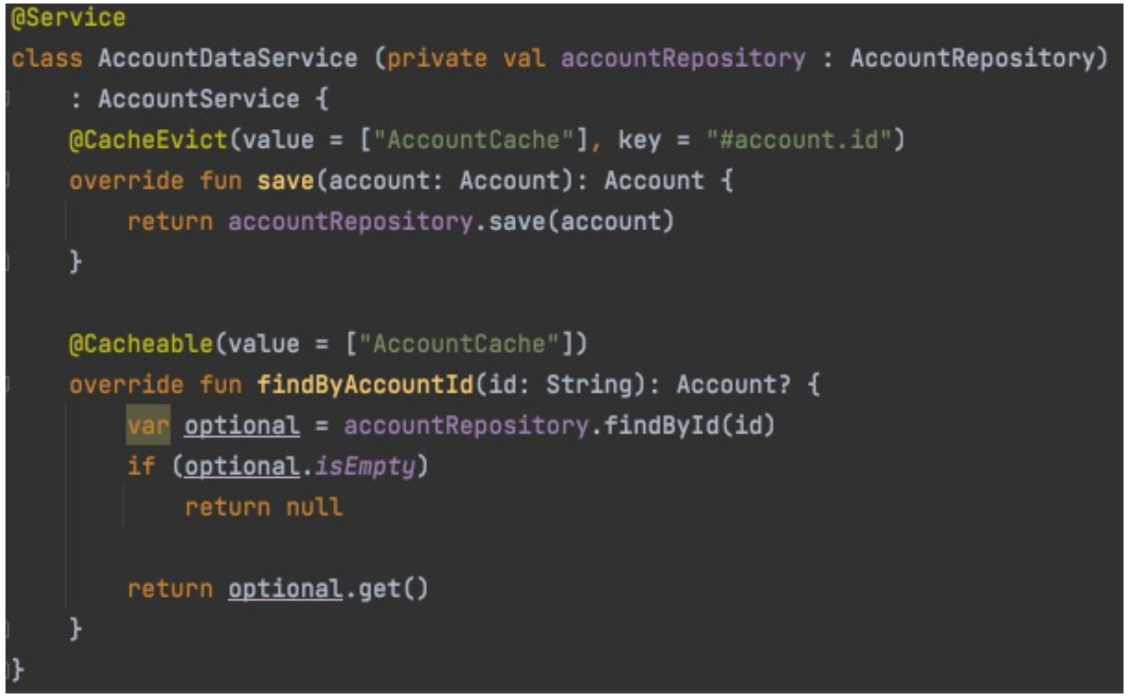
Java
Account REST Service
Kotlin
Acct Location Transaction Service
GemFire Cluster
Locator
Locator
Cache Server
Cache Server
Cache Server
Account Region PARTITIONED PERSISTENT
Location Region PARTITIONED PERSISTENT
WAN REPLICATION
GemFire Cluster
Locator
Locator
Cache Server
Cache Server
Cache Server
Account Region PARTITIONED PERSISTENT
Location Region PARTITIONED PERSISTENT
Kotlin
Acct Location Transaction Service

Cacheable

Look aside

- Use Case
  - Data can be loaded as needed
  - Data domains access by id
  - Need to minimize the initial cache storage needs
- Spring Cache Abstraction
  - **@Cacheable** - result is stored into the cache so on subsequent invocations (with the same arguments), the value in the cache is returned without having to execute the method.
  - **@CacheEvict -** perform cache *eviction*, that is methods that act as triggers for removing data from the cache.





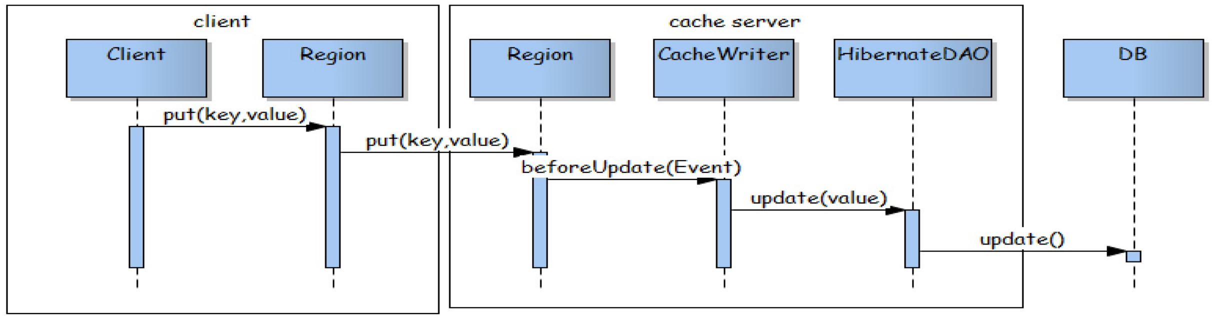
```
@Service
class AccountDataService (private val accountRepository : AccountRepository)
    : AccountService {
    @CacheEvict(value = ["AccountCache"], key = "#account.id")
    override fun save(account: Account): Account {
        return accountRepository.save(account)
    }

    @Cacheable(value = ["AccountCache"])
    override fun findByAccountId(id: String): Account? {
        var optional = accountRepository.findById(id)
        if (optional.isEmpty)
            return null

        return optional.get()
    }
}
```

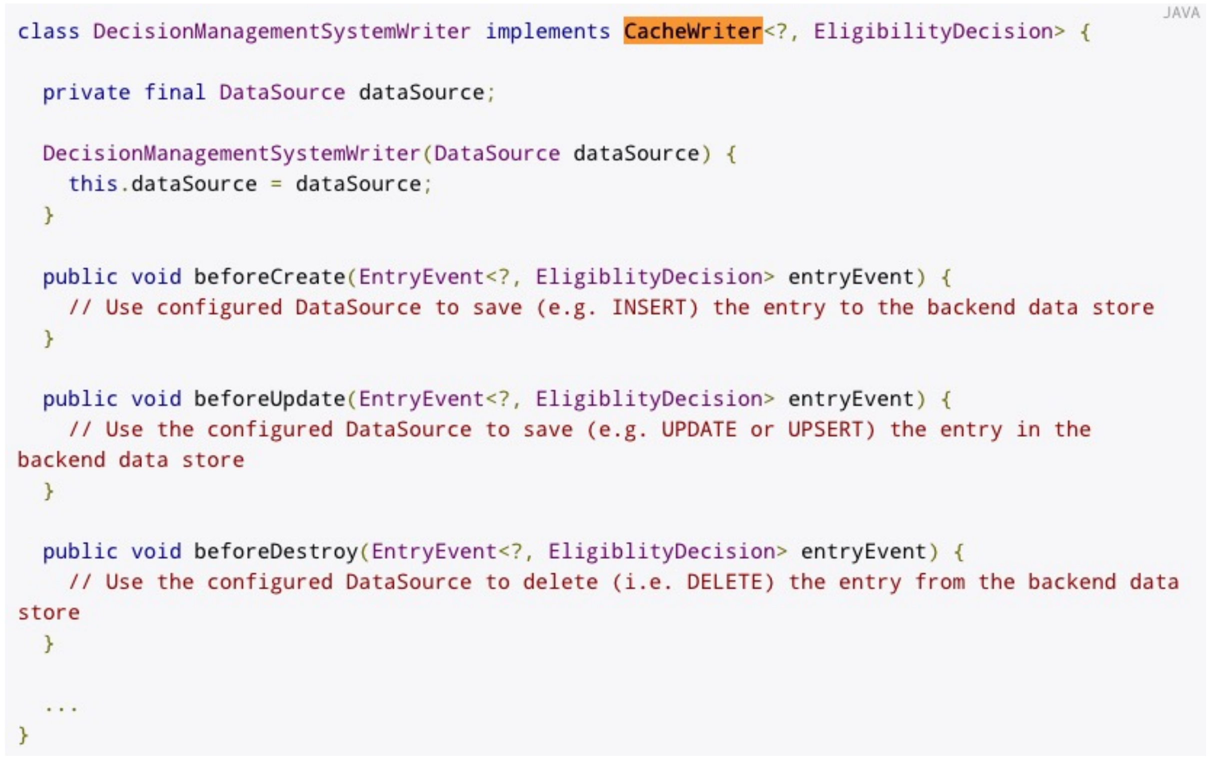
Cache Writer

Write through

- Use Case
  - Transactional data updates are required to maintain consistency between cache and external data systems
  - Real-time updates are required
  - Update/write rate relatively low
  - Cache write latency is acceptable

- Cache Writer
  - CacheWriters are user-defined and associated with a region.
  - Usage triggered by put key/value entry into a region.
  - CacheWriter’s beforeUpdate method may be called in the case of a put of existing record’s.
  - Their beforeUpdate, beforeCreate, beforeDestroy, beforeRegionClear, etc. methods are called synchronously before a region or entry in the cache is modified.
  - The region operation may be client and or server-side.



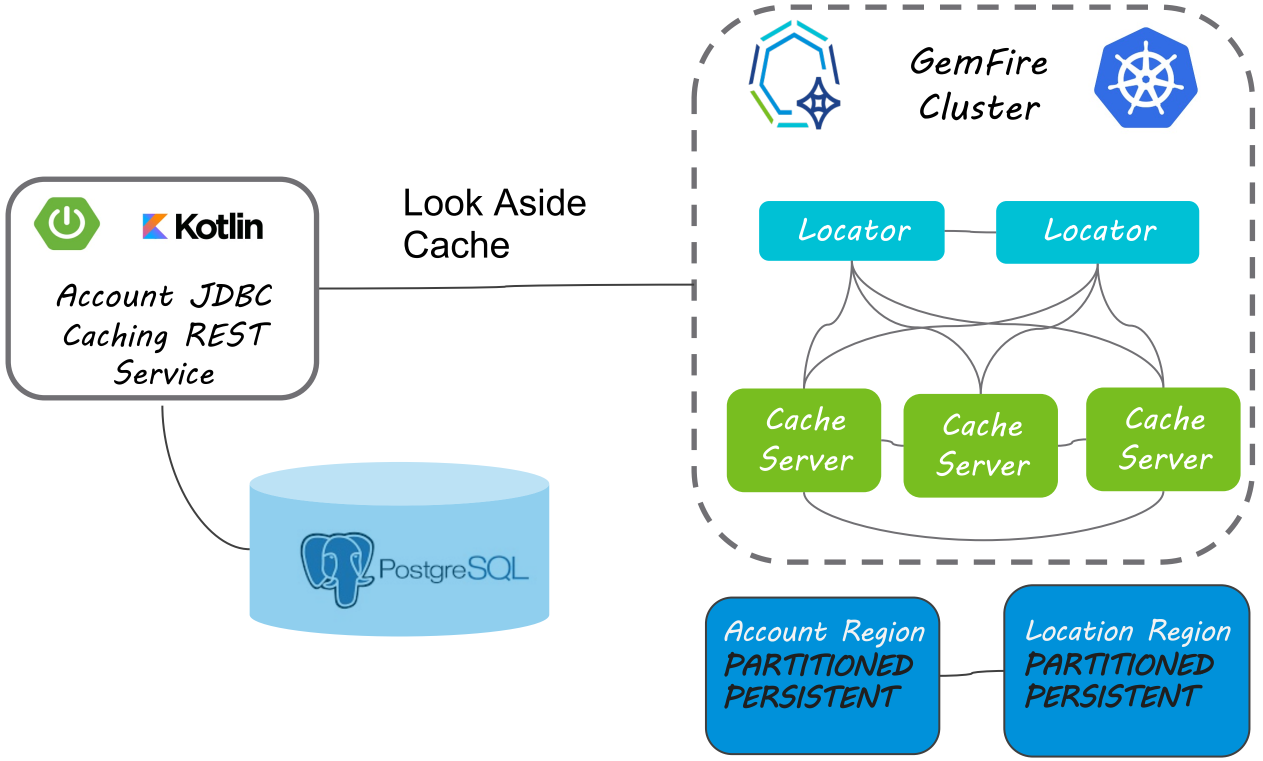


```java
class DecisionManagementSystemWriter implements CacheWriter<?, EligibilityDecision> {

  private final DataSource dataSource;

  DecisionManagementSystemWriter(DataSource dataSource) {
    this.dataSource = dataSource;
  }

  public void beforeCreate(EntryEvent<?, EligiblityDecision> entryEvent) {
    // Use configured DataSource to save (e.g. INSERT) the entry to the backend data store
  }

  public void beforeUpdate(EntryEvent<?, EligiblityDecision> entryEvent) {
    // Use the configured DataSource to save (e.g. UPDATE or UPSERT) the entry in the
backend data store
  }

  public void beforeDestroy(EntryEvent<?, EligiblityDecision> entryEvent) {
    // Use the configured DataSource to delete (i.e. DELETE) the entry from the backend data
store
  }

  ...
}
```

https://github.com/ggreen/spring-geode-showcase.git

Exercise

Look Aside Cache


Kotlin
Account JDBC Caching REST Service
Look Aside Cache
GemFire Cluster
Locator
Locator
Cache Server
Cache Server
Cache Server
PostgreSQL
Account Region PARTITIONED PERSISTENT
Location Region PARTITIONED PERSISTENT

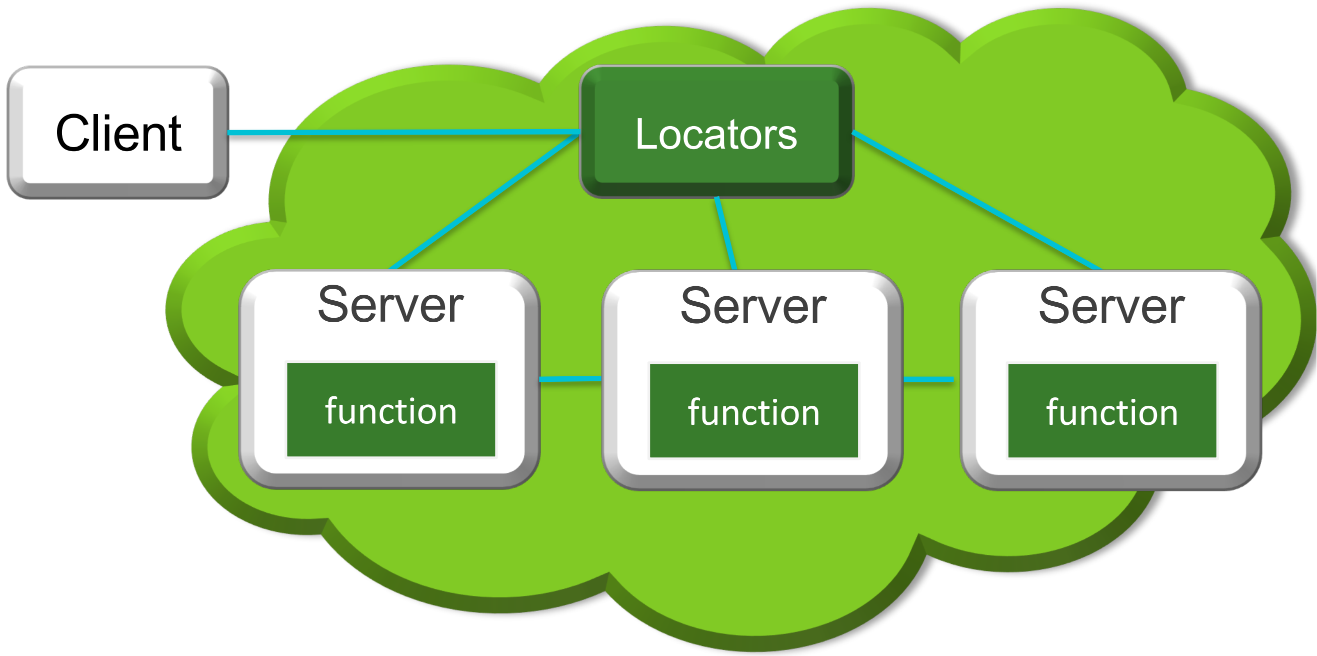
Functions

Functions are GemFire's equivalent to database stored procedures

- Execute business logic that is co-located with data in-memory
- Fastest data access patterns
- Functions can be made asynchronous by setting the *hasResult* flag to false and not returning a value.
- Function can be executed programmatically or manually through Gfsh

Execution Types

- **OnRegion** execute on region/partition
  - Executing code in the exact node where a specific key resides in a partitioned Region
- **OnServers** execute on all servers in a pool
  - Executing code simultaneously on all nodes
- **OnServer** execute on a single server in a pool
- **OnMember** execute on a particular server

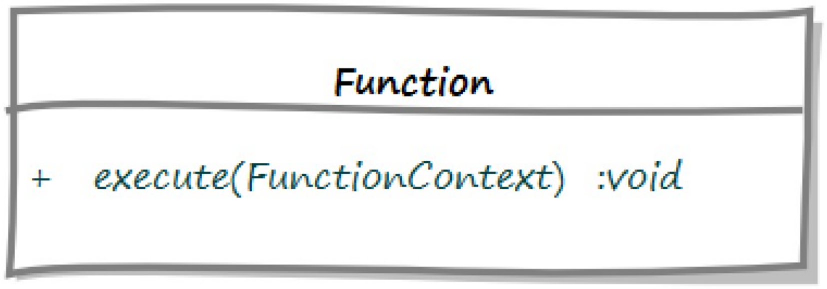


GemFire Functions

Register with XML

Command line or programmatic API

Execute through command line or programmatic API



```
<!- XML Registration →
<cache>
<function>
  <class-
name>demo.UserMgrOnRegionFunction</class-name>
</function>
</cache>
```

```
gfsh> deploy --jar=/tmp/myfunction.jar

gfsh> execute function --id=ClearRegionFunction
--region=/test
```

```
Execution execution = FunctionService.onRegion(exampleRegion)
    .withFilter(keysForGet)
    .setArguments(Boolean.TRUE)
    .withCollector(new MyArrayListResultCollector());

ResultCollector rc = execution.execute(function);
// Retrieve results, if the function returns results
List result = (List)rc.getResult();
```

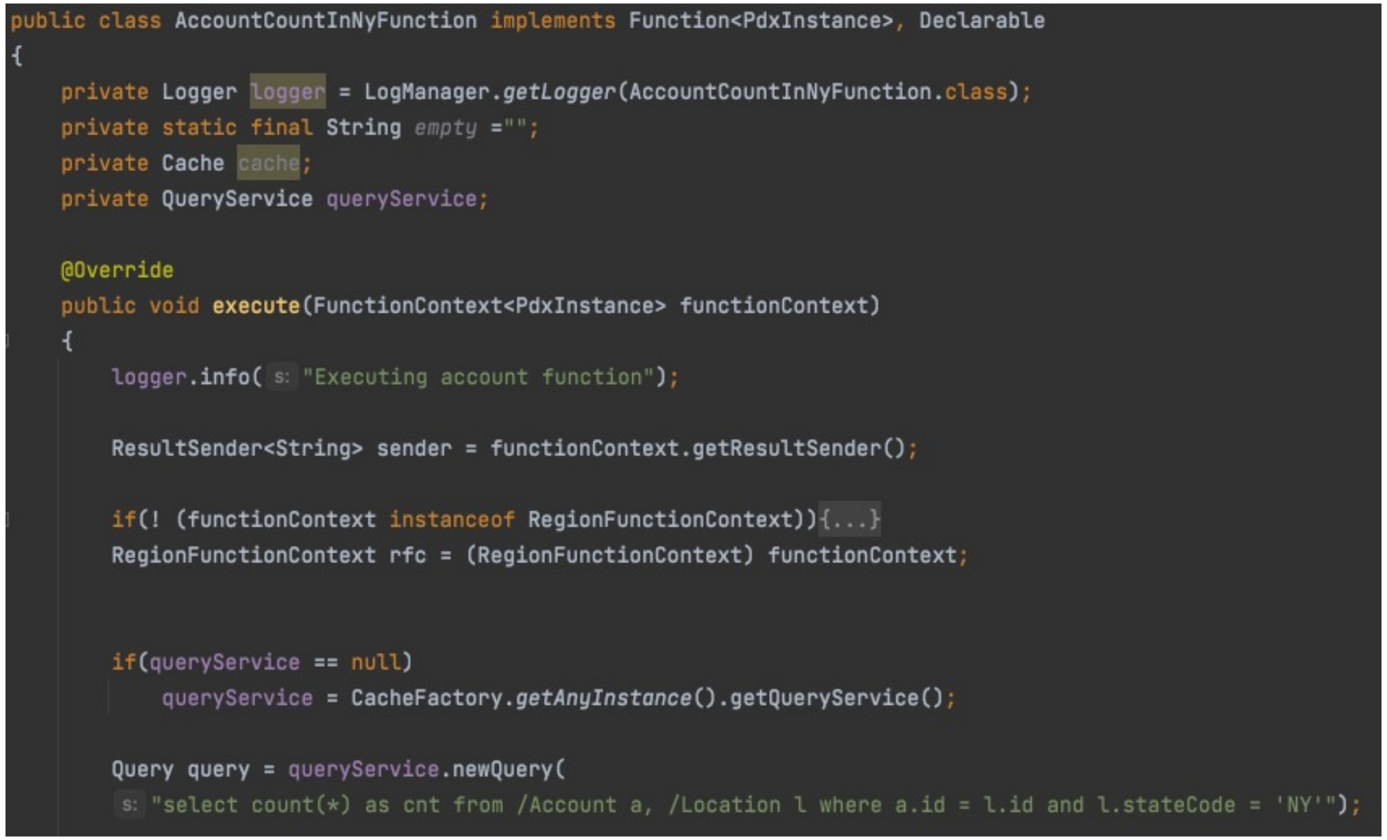
GemFire Functions Execution

Minimize network hops with functions

Co-locate data from different partitioned regions

- Group related partitioned region data on same member.
  - Improves queries and other operations that access data access time
  - Ex: Co-locate material, types, groups, units of measure, plants, MDM data based of accounts

```
create region --name=Account --type=PARTITION_PERSISTENT

create region --name=Location --type=PARTITION_PERSISTENT
--colocated-with=/Account
```

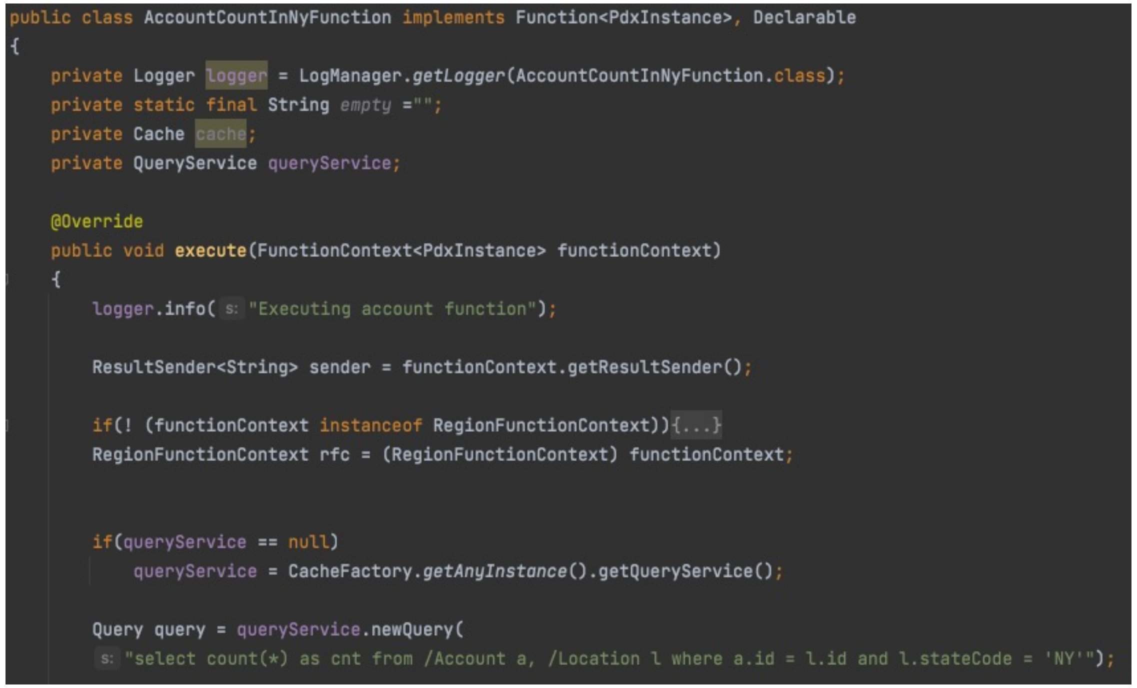


```
public class AccountCountInNyFunction implements Function<PdxInstance>, Declarable
{
    private Logger logger = LogManager.getLogger(AccountCountInNyFunction.class);
    private static final String empty ="";
    private Cache cache;
    private QueryService queryService;

    @Override
    public void execute(FunctionContext<PdxInstance> functionContext)
    {
        logger.info( s: "Executing account function");

        ResultSender<String> sender = functionContext.getResultSender();

        if(! (functionContext instanceof RegionFunctionContext)){...}
        RegionFunctionContext rfc = (RegionFunctionContext) functionContext;


        if(queryService == null)
            queryService = CacheFactory.getAnyInstance().getQueryService();

        Query query = queryService.newQuery(
         s: "select count(*) as cnt from /Account a, /Location l where a.id = l.id and l.stateCode = 'NY'");
```

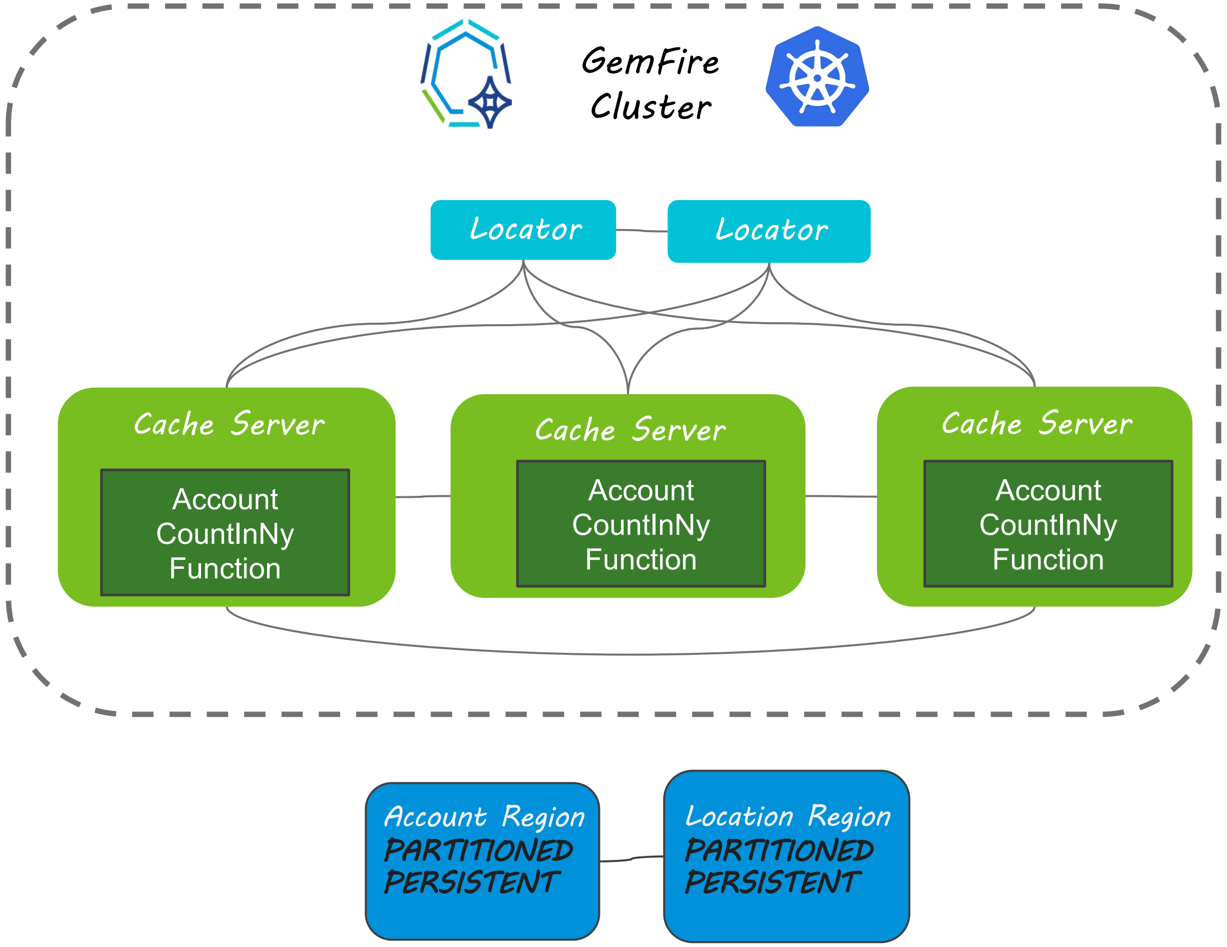
https://github.com/ggreen/spring-geode-showcase.git

Exercise

Function execution



```
public class AccountCountInNyFunction implements Function<PdxInstance>, Declarable
{
    private Logger logger = LogManager.getLogger(AccountCountInNyFunction.class);
    private static final String empty ="";
    private Cache cache;
    private QueryService queryService;

    @Override
    public void execute(FunctionContext<PdxInstance> functionContext)
    {
        logger.info( s: "Executing account function");

        ResultSender<String> sender = functionContext.getResultSender();

        if(! (functionContext instanceof RegionFunctionContext)){...}
        RegionFunctionContext rfc = (RegionFunctionContext) functionContext;


        if(queryService == null)
            queryService = CacheFactory.getAnyInstance().getQueryService();

        Query query = queryService.newQuery(
         s: "select count(*) as cnt from /Account a, /Location l where a.id = l.id and l.stateCode = 'NY'");
```


GemFire Cluster
Locator
Locator
Cache Server
Account CountInNy Function
Cache Server
Account CountInNy Function
Cache Server
Account CountInNy Function
Account Region PARTITIONED PERSISTENT
Location Region PARTITIONED PERSISTENT

https://github.com/ggreen/spring-geode-showcase.git

GemFire Development Links

- Core Java/Apache Blog
  - https://www.baeldung.com/apache-geode
- Spring Data Geode Blog
  - https://www.baeldung.com/spring-data-geode
- Tanzu GemFire Developer Center
  - https://tanzu.vmware.com/developer/data/tanzu-gemfire/

Thank You