Architecture of Enterprise Applications 21 Clustering

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - Overview of Clustering
  - Nginx
  - MySQL Clustering

- Objectives
  - 能够根据业务需求和系统被访问的模式，设计合理的集群部署方式和负载均衡策略

Clustering

- A large-scale system typically:
  - Has many user, potentially in many different places
  - Is long-running, that is, required to be “always up”
  - Processes large numbers of transactions per second
  - May see increases in both its user population and system load
  - Represents considerable business value
  - Is operated and managed by multiple persons

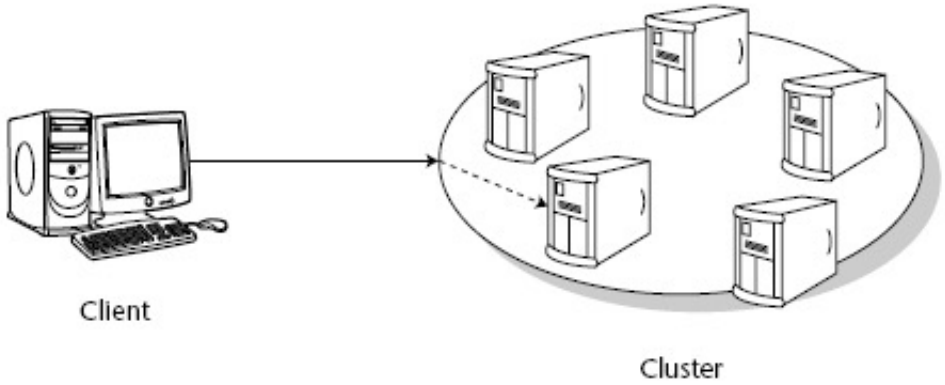
- Essential requirements on large-scale systems are often summarized by the following three properties(RAS):
  - Reliability
  - Availability
  - Serviceability
  - Scalability

Clustering

- Clustering addresses many of the issues faced by large-scale systems at the same time.
- A cluster is a loosely coupled group of servers that provide unified services to their clients.
- The client's view of the cluster is a single, simple system, not a group of collaborating servers. This is referred to as a single-system view or single-system image.
- Computers in a cluster are called nodes.

Clustering

- Clustering can be a very involved technology, potentially encompassing group communication and replication protocols, and network components such as load balancers and traffic redirectors at different layers in the protocol stack.



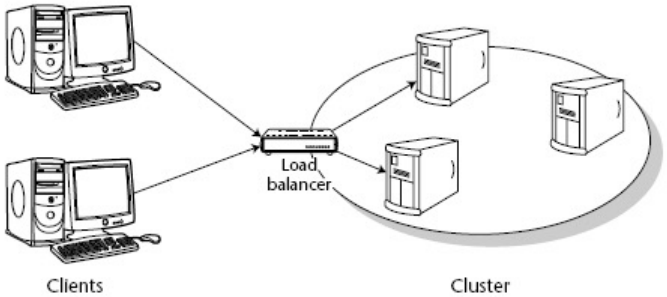
Clustering

- The main principle behind clustering is that of redundancy.
  - Reliability
    - Remove single points of failure
  - Availability
    - Overall availability is $1-(1-f\%)^n$
  - Serviceability
    - More complex than a single application server
    - But we could get ability for hot upgrade
  - Scalability
    - It is cheaper to build a cluster using standard hardware than to rely on multiprocessor machines.
    - Extending a cluster by adding extra servers can be done during operation and hence is less disruptive than plugging in another CPU board.

Load balancing and Failover

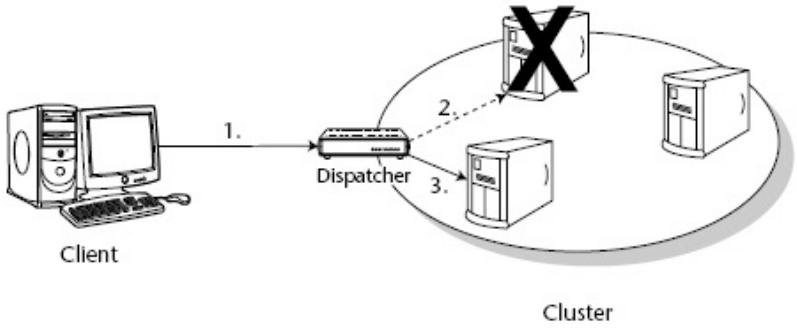
- Load balancing means distributing the requests among cluster nodes to optimize the performance of the whole system.
  - The algorithm that the load balancer uses to decide which target node to pick for a request can be systematic or random.
  - Alternatively, the load balancer could try to monitor the load on the different nodes in the cluster and pick node that appears less loaded than others.

- An important feature for Web load balancers is session stickiness, which means that all requests in a client's session are directed to the same server.



Load balancing and Failover

- For a cluster to provide higher availability to clients that a single server, the cluster must be able to failover from a primary server to another, secondary server when failures occur.
  - Request-level failover. It occurs when a request that is directed to one node for servicing cannot be serviced and is subsequently redirected to another node.
  - Session failover. If session state is shared between clients and servers, request-level failover may not be sufficient to continue operations. In this case, the session state must also be reconstructed at server node.



The concept of idempotence

- An idempotent method is one that can be called repeatedly with the same arguments and achieves the same results each time.
  - HTTP GET
  - Generally, any methods that alter a persistent store based on its current state are not idempotent, since two invocations of the same method will alter the persistent store twice.

- A failed request could have occurred at one of three points:
  - After the request has been initiated but before method invocation on the server has begun to execute.
  - After the method invocation on the server has begun to execute, but before the method has completed.
  - After the method invocation on the server has completed but before the response has been successfully transmitted to the remote client.

- nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server, written by Igor Sysoev.
  - For a long time, it has been running on many heavily loaded Russian sites including
    - Yandex, Mail.Ru, VKontakte, and Rambler.

- According to Netcraft nginx served or proxied 25.68% busiest sites in February 2020.
  - Here are some of the success stories:
    - Dropbox, Netflix, Wordpress.com, FastMail.FM.

- Starting, Stopping, and Reloading Configuration
  - To start nginx, run the executable file.
  - Once nginx is started, it can be controlled by invoking the executable with the `-s` parameter.
  - Use the following syntax:
  - `nginx -s signal` Where *`signal`* may be one of the following:
    - `stop` — fast shutdown
    - `quit` — graceful shutdown
    - `reload` — reloading the configuration file
    - `reopen` — reopening the log files

- Serving Static Content
  - First, create the `/data/www` directory and put an `index.html` file with any text content into it and create the `/data/images` directory and place some images in it.
  - Next, open the configuration file. The default configuration file already includes several examples of the `server` block, mostly commented out.

```
server {
 location / {
   root /data/www;
 }
 location /images/ {
   root /data;
 }
}
```

- Serving Static Content
  - This is already a working configuration of a server that listens on the standard port 80 and is accessible on the local machine at `http://localhost/.`
  - In response to requests with URIs starting with /images/, the server will send files from the /data/images directory.
    - For example, in response to the `http://localhost/images/example.png` request nginx will send the `/data/images/example.png` file. If such file does not exist, nginx will send a response indicating the 404 error.
  - Requests with URIs not starting with `/images/` will be mapped onto the `/data/www` directory.
    - For example, in response to the `http://localhost/some/example.html` request nginx will send the `/data/www/some/example.html` file.

- Setting Up a Simple Proxy Server
  - The configuration of a proxy server will look like this:

```
server {
 location / {
  proxy_pass http://localhost:8080/;
 }
 location ~ \.(gif|jpg|png)$ {
  root /data/images;
 }
}
```

  - This server will filter requests ending with .gif, .jpg, or .png and map them to the `/data/images` directory (by adding URI to the root directive's parameter) and pass all other requests to the proxied server configured above.

- Load balancing methods
  - The following load balancing mechanisms (or methods) are supported in nginx:
    - round-robin — requests to the application servers are distributed in a round-robin fashion,
    - least-connected — next request is assigned to the server with the least number of active connections,
    - ip-hash — a hash-function is used to determine what server should be selected for the next request (based on the client's IP address).

- Default load balancing configuration
  - The simplest configuration for load balancing with nginx may look like the following:

```
http {
 upstream myapp1 {
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

  - Reverse proxy implementation in nginx includes load balancing for HTTP, HTTPS, FastCGI, uwsgi, SCGI, and memcached.

- Least connected load balancing

```
http {
 upstream myapp1 {
  least_conn;
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

- Session persistence
  - Please note that with round-robin or least-connected load balancing, each subsequent client's request can be potentially distributed to a different server.
    - There is no guarantee that the same client will be always directed to the same server.
  - If there is the need to tie a client to a particular application server
    - in other words, make the client's session "sticky" or "persistent" in terms of always trying to select a particular server — the ip-hash load balancing mechanism can be used.

```
http {
 upstream myapp1 {
  ip_hash;
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

- Weighted load balancing

```
http {
 upstream myapp1 {
  server srv1.example.com weight=3;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

Install and Run Nginx

- Mac OS
  - $ sudo chown -R `whoami` /usr/local/Homebrew/
  - $ sudo chown -R $(whoami) $(brew --prefix)/*
  - $ sudo mkdir /usr/local/Frameworks
  - $ sudo chown -R `whoami` /usr/local/Frameworks/
  - $ brew install nginx
  - $ nginx
- Windows
  - Download and Unzip the stable version of Nginx on http://nginx.org/en/download.html
  - Run nginx.exe

Configuration of Nginx

- Mac OS
  - $ cd /usr/local/etc/nginx/
  - Edit nginx.conf
  - nginx
  - nginx -s stop / reopen
- Windows
  - cd %NGINX_HOME%/conf
  - Edit nginx.conf

```
http {
   upstream pancm{
    #ip_hash;
    #least_conn;
    server 127.0.0.1:8080; #weight=3;
    server 127.0.0.1:8090;
   }
   server {
      listen       8000;
      server_name  localhost;
      location / {
         root   html;
         proxy_pass http://pancm;
         index  index.html index.htm;
      }
```

Prepare Two Spring Boot Projects

- ServerOne
  - GreetingController.java

```
@CrossOrigin(maxAge = 3600)
@RestController
public class GreetingController {
  @GetMapping("/")
  public String getHome() {
    return "Let' start! Server One";
  }

}
```

  - application.properties

```
server.port=8080
```

- ServerTwo
  - GreetingController.java

```
@CrossOrigin(maxAge = 3600)
@RestController
public class GreetingController {
  @GetMapping("/")
  public String getHome() {
    return "Let' start! Server Two";
  }

}
```

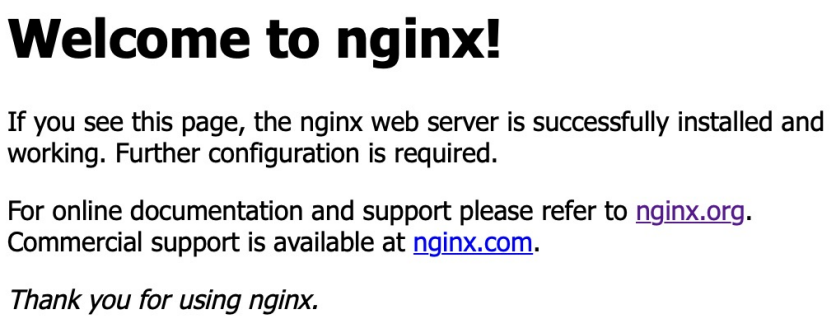
  - application.properties

```
server.port=8090
```

Test Nginx

REliable, INtelligent & Scalable Systems

- Run Nginx
  - http://localhost:8000



- Run the two web modules with the following nginx configurations in turn.
  - default(round-robin)
  - ip_hash
  - least_conn
  - With different weights

http://localhost:8000

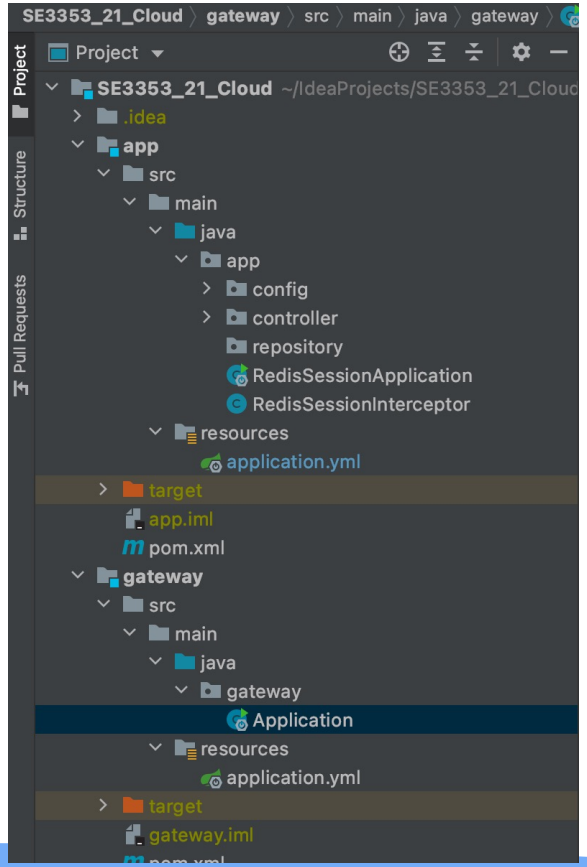
Let' start! Server One

Let' start! Server Two

Clustering with Redis for Sessions

REliable, INtelligent & Scalable Systems

- Getway Module
  - Run at 8001 port and route request
- App Module
  - Run at 8002 port and process request
- Redis
  - Run at 6379 port and store sessions



Clustering with Redis for Sessions

- Application.java

```
package gateway;

@SpringBootApplication
public class Application {
   @Bean
   public RouteLocator myRoutes(RouteLocatorBuilder builder) {
      return builder.routes()
            .route(p->p.path("/login").filters(f -> f.rewritePath("/login", "/session/login")).uri("http://localhost:8002"))
            .route(p->p.path("/logout").filters(f->f.rewritePath("/logout", "/session/logout")).uri("http://localhost:8002"))
            .route(p->p.path("/getInfo").filters(f->f.rewritePath("/getInfo", "/session/getInfo")).uri("http://localhost:8002"))
            .build();
   }

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

- Application.yml

```
server:
  port: 8002
```

Clustering with Redis for Sessions

- RedisSessionApplication.java

```
@RestController
@SpringBootApplication
public class RedisSessionApplication {
   public static void main(String[] args) {
      SpringApplication.run(RedisSessionApplication.class, args);
   }
}
```

- Application.yml

```
server:
  port: 8002
spring:
  application:
    name: service-session-redis
  session:
    store-type: redis
    redis:
      namespace: spring:session
  redis:
    host: localhost
    port: 6379
    password:
    timeout: 6000
    jedis:
      pool:
        max-active: 8
        max-wait: -1ms
        max-idle: 8
        min-idle: 0
    database: 0
management:
  endpoints:
    web:
      exposure:
        include: "*"
      cors:
        allowed-origins: "*"
        allowed-methods: "*"
```

Clustering with Redis for Sessions

- RedisConfig.java

```
@Configuration
public class RedisConfig extends CachingConfigurerSupport {
   @Value("${spring.redis.host}")
   private String host;
   @Value("${spring.redis.port}")
   private int port;
   @Value("${spring.redis.password}")
   private String password;
   @Value("${spring.redis.timeout}")
   private int timeout;

   @Bean
   public JedisPool redisPoolFactory() {
      JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
      return new JedisPool(jedisPoolConfig, host, port, timeout, password);
   }
```

Clustering with Redis for Sessions

REliable, INtelligent & Scalable Systems

- RedisConfig.java

```
@Bean(name = "redisTemplate")
public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory){
    RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
    redisTemplate.setConnectionFactory(redisConnectionFactory);

    ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
    objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);

    Jackson2JsonRedisSerializer<Object> jsonRedisSerializer = new Jackson2JsonRedisSerializer<>(Object.class);
    jsonRedisSerializer.setObjectMapper(objectMapper);
    redisTemplate.setDefaultSerializer(jsonRedisSerializer);

    redisTemplate.setKeySerializer(new StringRedisSerializer());
    redisTemplate.afterPropertiesSet();
    return redisTemplate;
  }
}
```

Clustering with Redis for Sessions

- RedisSessionConfig.java

```
@Configuration
@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 60)
public class RedisSessionConfig {
}
```

- WebSecurityConfig.java

```
@Configuration
public class WebSecurityConfig extends WebMvcConfigurerAdapter {
   @Autowired
   RedisSessionInterceptor redisSessionInterceptor;

   @Override
   public void addInterceptors(InterceptorRegistry registry){
      registry.addInterceptor(redisSessionInterceptor)
            .addPathPatterns("/session/**")
            .excludePathPatterns("/session/login");
      super.addInterceptors(registry);
   }
}
```

Clustering with Redis for Sessions

- userManagementController.java

```
@RestController
public class userManagementController {
    private StringRedisTemplate redisTemplate;
    @Autowired
    public userManagementController(StringRedisTemplate redisTemplate){
        this.redisTemplate = redisTemplate;
    }

    @GetMapping("/session/login")
    public String login(HttpServletRequest request){
        HttpSession session = request.getSession();
        session.setAttribute("username", "zzt");
        redisTemplate.opsForValue().set("username:zzt", session.getId());
        return "login finished";
    }

    @GetMapping("/session/logout")
    public String logout(HttpSession session){
        session.invalidate();
        return "logout finished";
    }
```

Clustering with Redis for Sessions

- userManagementController.java

```
@GetMapping(value = "/session/getInfo")
public Map<String, String> addSession (HttpServletRequest request){
    HttpSession session = request.getSession();
    String sessionId = session.getId();
    String username = (String) session.getAttribute("username");
    String requestURI = request.getRequestURI();


    Map<String, String> sessionInfoMap = new HashMap<>();
    sessionInfoMap.put("sessionId", sessionId);
    sessionInfoMap.put("username", username);
    sessionInfoMap.put("requestURI", requestURI);
    return sessionInfoMap;
  }
}
```

Clustering with Redis for Sessions

REliable, INtelligent & Scalable Systems

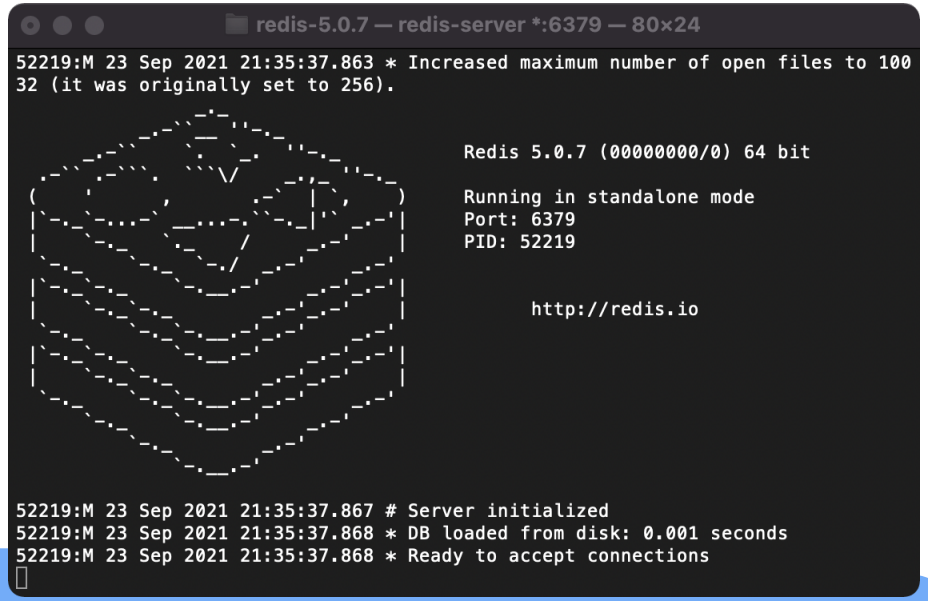
- RedisSessionInterceptor.java

```
@Slf4j
@Component
public class RedisSessionInterceptor implements HandlerInterceptor {
   private final StringRedisTemplate redisTemplate;

   @Autowired
   public RedisSessionInterceptor(StringRedisTemplate redisTemplate){
      this.redisTemplate = redisTemplate;
   }

   @Override
   public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception{
      HttpSession session = request.getSession();
      String username = (String) session.getAttribute("username");
      if(username != null){
         String loginSessionId = redisTemplate.opsForValue().get(String.format("username:%s", username));
         if(loginSessionId != null && loginSessionId.equals(session.getId())){
            // success
            return true;
         }else if(loginSessionId != null && !loginSessionId.equals(session.getId())){
            log.info("id mismatch, session invalidate: {}", session.getId());
         }
      }else{
         log.info("please login");
      }
      response401(response);
      return false;
   }
```

Clustering with Redis for Sessions

- RedisSessionInterceptor.java

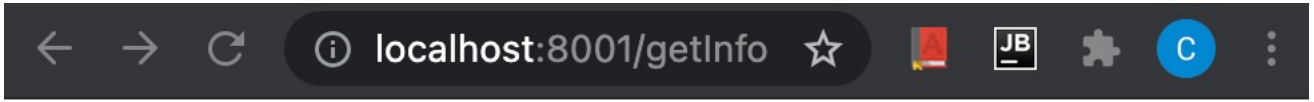
```
    private void response401(HttpServletResponse response) throws Exception{
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json; charset=utf-8");
        response.getWriter().print("401");
    }
}
```



Clustering with Redis for Sessions

401

login finished

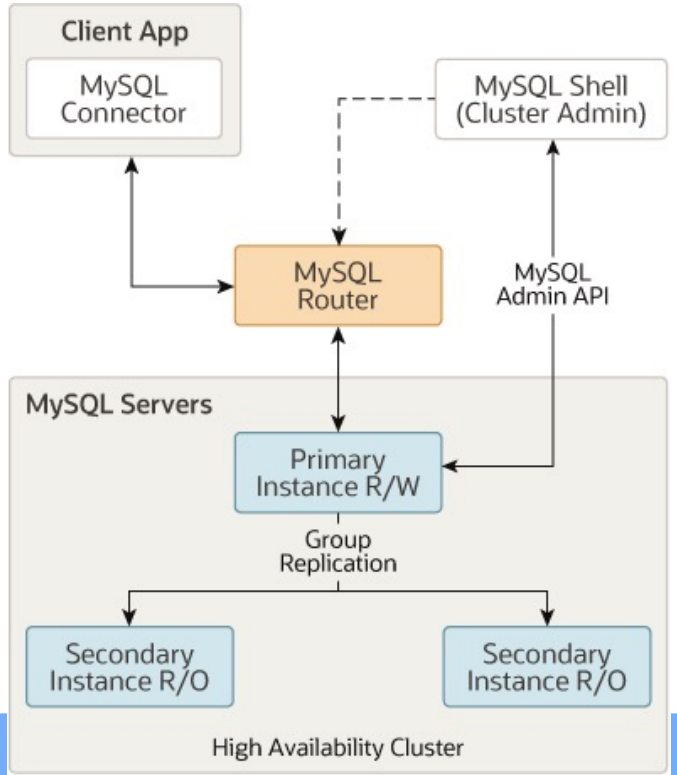


{"sessionId":"010d4cbd-e52e-43aa-86fe-8d45dc21ef51","requestURI":"/session/getInfo","username":"zzt"}

```
127.0.0.1:6379> KEYS spring*
1) "spring:session:sessions:010d4cbd-e52e-43aa-86fe-8d45dc21ef51"
2) "spring:session:expirations:1632404520000"
```

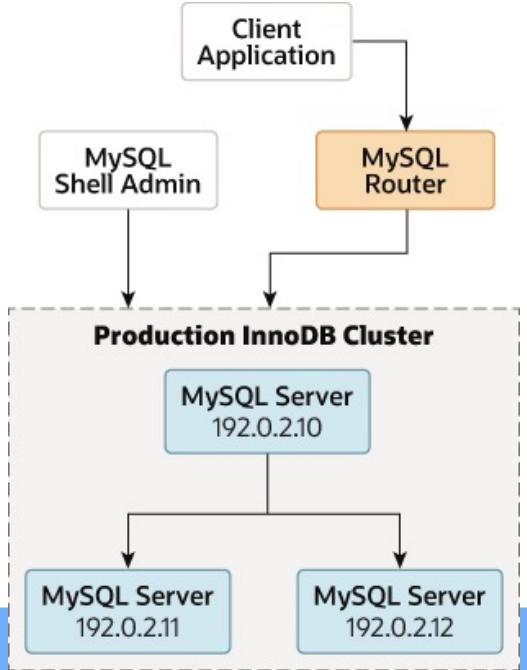
MySQL InnoDB Cluster

- MySQL InnoDB Cluster provides a complete high availability solution for MySQL.



Deploying a Production InnoDB Cluster

- When working in a production environment,
  - the MySQL server instances which make up an InnoDB Cluster run on multiple host machines as part of a network rather than on single machine.



Deploying a New Production InnoDB Cluster

- **Configuring Production Instances**
  - AdminAPI provides the dba.configureInstance() function that checks if an instance is suitably configured for InnoDB Cluster usage, and configures the instance if it finds any settings which are not compatible with InnoDB Cluster.

    dba.configureInstance([*instance*][, *options*])

  - for example:

    dba.configureInstance(*user@example*:*3306*)

  - A report is displayed which shows the settings required by InnoDB Cluster .

Creating a Cluster

- Once you have prepared your instances,
  - use the dba.createCluster() function to create the cluster,
  - using the instance which MySQL Shell is connected to as the seed instance for the cluster.
  - The seed instance is replicated to the other instances that you add to the cluster, making them replicas of the seed instance.

```
mysql-js> var cluster = dba.createCluster('testCluster')
Validating instance at icadmin@ic-1:3306...
This instance reports its own address as ic-1
Instance configuration is suitable.
Creating InnoDB cluster 'testCluster' on 'icadmin@ic-1:3306'...
Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to one server failure.
```

Adding instance to a Cluster

- Use the *Cluster*.addInstance(*instance*) function to add more instances to the cluster
  - You need a minimum of three instances in the cluster to make it tolerant to the failure of one instance.
  - Adding further instances increases the tolerance to failure of an instance.

```
mysql-js> cluster.addInstance('icadmin@ic-2:3306')
A new instance will be added to the InnoDB cluster. Depending on the amount of data on the cluster this might take from a few seconds to several hours.
Please provide the password for 'icadmin@ic-2:3306': ********
Adding instance to the cluster ...
Validating instance at ic-2:3306...
This instance reports its own address as ic-2 Instance configuration is suitable.
The instance 'icadmin@ic-2:3306' was successfully added to the cluster.
```

Adding instance to a Cluster

- If you are using MySQL 8.0.17 or later
  - you can choose how the instance recovers the transactions it requires to synchronize with the cluster.
  - Only when the joining instance has recovered all of the transactions previously processed by the cluster can it then join as an online instance and begin processing transactions.

- Depending on which option you chose to recover the instance from the cluster,
  - you see different output in MySQL Shell.
  - Suppose that you are adding the instance ic-2 to the cluster, and ic-1 is the seed or donor.

Adding instance to a Cluster

- When you use MySQL Clone to recover an instance from the cluster, the output looks like:

```
Validating instance at ic-2:3306...
This instance reports its own address as ic-2:3306
Instance configuration is suitable.
A new instance will be added to the InnoDB cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.
Adding instance to the cluster...
Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.
NOTE: A server restart is expected to happen as part of the clone process. If the
server does not support the RESTART command or does not come back after a
while, you may need to manually start it back.
* Waiting for clone to finish...
NOTE: ic-2:3306 is being cloned from ic-1:3306
** Stage DROP DATA: Completed
** Clone Transfer
FILE COPY  ############################################################  100%  Completed
PAGE COPY  ############################################################  100%  Completed
REDO COPY  ############################################################  100%  Completed
NOTE: ic-2:3306 is shutting down...
* Waiting for server restart... ready
* ic-2:3306 has restarted, waiting for clone to finish...
** Stage RESTART: Completed
* Clone process has finished: 2.18 GB transferred in 7 sec (311.26 MB/s)
State recovery already finished for 'ic-2:3306'
The instance 'ic-2:3306' was successfully added to the cluster.
```

Adding instance to a Cluster

- When you use incremental recovery to recover an instance from the cluster, the output looks like:

```
Incremental distributed state recovery is now in progress.
* Waiting for incremental recovery to finish...
NOTE: 'ic-2:3306' is being recovered from 'ic-1:3306'
* Distributed recovery has finished
```

Adding instance to a Cluster

- To verify the instance has been added, use the cluster instance's status() function.
  - For example this is the status output of a sandbox cluster after adding a second instance:

```
mysql-js> cluster.status()
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ic-1:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "ic-1:3306": {
                "address": "ic-1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-2:3306": {
                "address": "ic-2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    },
    "groupInformationSourceMember": "mysql://icadmin@ic-1:3306"
}
```

Monitoring InnoDB Cluster

- Using *Cluster*.describe()

```
mysql-js> cluster.describe();
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "topology": [
            {
                "address": "ic-1:3306",
                "label": "ic-1:3306",
                "role": "HA"
            },
            {
                "address": "ic-2:3306",
                "label": "ic-2:3306",
                "role": "HA"
            },
            {
                "address": "ic-3:3306",
                "label": "ic-3:3306",
                "role": "HA"
            }
        ]
    }
}
```

Monitoring InnoDB Cluster

REliable, INtelligent & Scalable Systems

- Checking a cluster's Status with *Cluster*.status()

```
mysql-js> var cluster = dba.getCluster()
mysql-js> cluster.status()
{
    "clusterName": "testcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ic-1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "ic-1:3306": {
                "address": "ic-1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-2:3306": {
                "address": "ic-2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-3:3306": {
                "address": "ic-3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    },
    "groupInformationSourceMember": "mysql://icadmin@ic-1:3306"
}
```

Working with Instances

REliable, INtelligent & Scalable Systems

- Using dba.checkInstanceConfiguration()

```
mysql-js> dba.checkInstanceConfiguration('icadmin@ic-1:3306')
Please provide the password for 'icadmin@ic-1:3306': ***
Validating MySQL instance at ic-1:3306 for use in an InnoDB cluster...

This instance reports its own address as ic-1
Clients and other cluster members will communicate with it through this address by default.
If this is not correct, the report_host MySQL system variable should be changed.

Checking whether existing tables comply with Group Replication requirements...
No incompatible tables detected

Checking instance configuration...

Some configuration options need to be fixed:
+--------------------------+---------------+----------------+--------------------------------------------------+
| Variable                 | Current Value | Required Value | Note                                             |
+--------------------------+---------------+----------------+--------------------------------------------------+
| enforce_gtid_consistency | OFF           | ON             | Update read-only variable and restart the server |
| gtid_mode                | OFF           | ON             | Update read-only variable and restart the server |
| server_id                | 1             |                | Update read-only variable and restart the server |
+--------------------------+---------------+----------------+--------------------------------------------------+
```

Working with Instances

REliable, INtelligent & Scalable Systems

- Using dba.checkInstanceConfiguration()

```
Please use the dba.configureInstance() command to repair these issues.

{
    "config_errors": [
        {
            "action": "restart",
            "current": "OFF",
            "option": "enforce_gtid_consistency",
            "required": "ON"
        },
        {
            "action": "restart",
            "current": "OFF",
            "option": "gtid_mode",
            "required": "ON"
        },
        {
            "action": "restart",
            "current": "1",
            "option": "server_id",
            "required": ""
        }
    ],
    "status": "error"
}
```

Working with Instances

- Configuring Instances with dba.configureLocalInstance()
  - If you issue dba.configureLocalInstance(), it enables MySQL Shell to modify the instance's option file after running the following commands against a remote instance:
  - dba.configureInstance()
  - dba.createCluster()
  - *Cluster*.addInstance()
  - *Cluster*.removeInstance()
  - *Cluster*.rejoinInstance()

Working with Instances

- Checking Instance State
  - The cluster.checkInstanceState() function can be used to verify the existing data on an instance does not prevent it from joining a cluster.

- The output of this function can be one of the following:
  - OK new: the instance has not executed any GTID transactions, therefore it cannot conflict with the GTIDs executed by the cluster
  - OK recoverable: the instance has executed GTIDs which do not conflict with the executed GTIDs of the cluster seed instances
  - ERROR diverged: the instance has executed GTIDs which diverge with the executed GTIDs of the cluster seed instances
  - ERROR lost_transactions: the instance has more executed GTIDs than the executed GTIDs of the cluster seed instances

Working with InnoDB Cluster

- Removing Instances from the InnoDB Cluster

```
mysql-js> cluster.removeInstance('root@localhost:3310')

The instance will be removed from the InnoDB cluster. Depending on the instance
being the Seed or not, the Metadata session might become invalid. If so, please
start a new session to the Metadata Storage R/W instance.

Attempting to leave from the Group Replication group...

The instance 'localhost:3310' was successfully removed from the cluster.
```

- Dissolving an InnoDB Cluster
  - The *Cluster*.dissolve() operation can only configure instances which are ONLINE or reachable. If members of a cluster cannot be reached by the member where you issued the *Cluster*.dissolve() command you have to decide how the dissolve operation should proceed.

```
mysql-js> Cluster.dissolve({force: true})
```

```
mysql-js> Cluster.dissolve({interactive: true})
```

- Changing a Cluster's Topology
  - *Cluster*.switchToMultiPrimaryMode(), which switches the cluster to multi-primary mode. All instances become primaries.
  - *Cluster*.switchToSinglePrimaryMode([*instance*]), which switches the cluster to single-primary mode.
    - If *instance* is specified, it becomes the primary and all the other instances become secondaries.
    - If *instance* is not specified, the new primary is the instance with the highest member weight (and the lowest UUID in case of a tie on member weight).

- Changing a Cluster's Topology
  - By default,
    - an InnoDB Cluster runs in single-primary mode, where the cluster has one primary server that accepts read and write queries (R/W),
    - and all of the remaining instances in the cluster accept only read queries (R/O).
    - When you configure a cluster to run in multi-primary mode, all of the instances in the cluster are primaries, which means that they accept both read and write queries (R/W).
    - If a cluster has all of its instances running MySQL server version 8.0.15 or later, you can make changes to the topology of the cluster while the cluster is online.
    - In previous versions it was necessary to completely dissolve and re-create the cluster to make the configuration changes.

Configuring InnoDB Cluster

- You can check and modify the settings in place for an InnoDB Cluster while the instances are online.

- To check the current settings of a cluster, use the following operation:
  - *Cluster*.options(), which lists the configuration options for the cluster and its instances.
    - A boolean option all can also be specified to include information about all Group Replication system variables in the output.
  - *Cluster*.setOption(*option*, *value*) to change the settings of all cluster instances globally or cluster global settings such as clusterName.
  - *Cluster*.setInstanceOption(instance, *option*, *value*) to change the settings of individual cluster instances

Configuring InnoDB Cluster

- These options are changeable at both the cluster (all instances) and per instance level:
  - autoRejoinTries: integer value to define the number of times an instance attempts to rejoin the cluster after being expelled.
  - exitStateAction: string value indicating the Group Replication exit state action.
  - memberWeight: integer value with a percentage weight for automatic primary election on failover.
  - tag:*option*: built-in and user-defined tags to be associated to the cluster.

Configuring InnoDB Cluster

- These options are changeable at the cluster level only:
  - clusterName: string value to define the cluster name
  - disableClone: boolean value used to disable the clone usage on the cluster.
  - expelTimeout: integer value to define the time period in seconds that cluster members should wait for a non-responding member before evicting it from the cluster.
  - failoverConsistency: string value indicating the consistency guarantees that the cluster provides.

- This option is changeable at the per instance level only:
  - label: a string identifier of the instance

Configuring InnoDB Cluster

- Customizing InnoDB Clusters
  - To customize the name of the replication group created by InnoDB Cluster,
    - pass the groupName option to the dba.createCluster() command.
  - To customize the address which an instance provides for connections from other instances,
    - pass the localAddress option to the dba.createCluster() and cluster.addInstance() commands.
  - To customize the instances used as seeds when an instance joins the cluster,
    - pass the groupSeeds option to the dba.createCluster() and *Cluster*.addInstance() operations.

Configuring InnoDB Cluster

- Configuring the Election Process
  - Use the memberWeight option and pass it to the dba.createCluster() and Cluster.addInstance() methods when creating your cluster.
  - The memberWeight option accepts an integer value between 0 and 100, which is a percentage weight for automatic primary election on failover.

```
dba.createCluster('cluster1', {memberWeight:35})
var mycluster = dba.getCluster()
mycluster.addInstance('icadmin@ic2', {memberWeight:25})
mycluster.addInstance('icadmin@ic3', {memberWeight:50})
```

作业十(第一部分)

- 请你根据上课内容，针对你在E-BookStore项目中的数据库设计，完成下列任务：
  1. 请你参照课程样例，构建你的E-BookStore的集群，它应该至少包含 1 个nginx实例(负载均衡) + 1 个Redis实例(存储session) + 2 个Tomcat实例 + 2个MySQL实例(主从备份)。(4分)
  2. 所使用的框架不限，例如可以不使用nginx而选用其他负载均衡器，或不使用Redis而选用其他缓存工具。

  - 请提交一份Word文档，详细叙述你的实现方式；并提交你的工程代码。

- 评分标准：
  - 能够正确地部署和运行上述系统，在迭代验收时需当面演示。
  - 部署方案不满足条件或无法正确运行，则视情况扣分。

References

- Nginx
  - http://nginx.org/en/docs/beginners_guide.html
- 完美解决Error: Running Homebrew as root is extremely dangerous and no longer supported.
  - https://blog.csdn.net/meifannao789456/article/details/105083605
- Spring boot集群 + Nginx
  - https://blog.csdn.net/chali1314/article/details/113310981
- 启动tomcat: Permission denied错误
  - https://blog.csdn.net/Pompeii/article/details/40262785
- MySQL InnoDB Cluster
  - https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-innodb-cluster.html

Thank You!