

Livy Server discovery

Part 1: Functional Specification	1
Background	1
Rationale	1
Part 2: Technical Specification	2
Solution. Livy Server discovering via ZooKeeper	2
ZooKeeper as backend store for Livy address	2
Implementation	2
Main idea and architecture	2
Detailed implementation	4
Configuration	4
Client code examples	5

Part 1: Functional Specification

Background

In this specification, we will describe technical details for Livy Server discovering. The feature which provides ability to get Livy Server address via API instead of configuring it explicitly by **livy.server.host** or **zeppelin.livy.url** (as we do for Zeppelin integration).

Rationale

As a developer I want to use Livy to execute some Spark code and provide REST API for my service for it. I need to communicate with Livy Server via HTTP from my service. To be able to use Livy REST interface I need to know **host** and **port** of Livy server. Typically we can use **livy.server.host** property to get Livy Server host address. This property should be configured in *livy.conf* before starting Livy. But usually we don't know exactly where Livy will be started. And we need to change this configuration when we changing Livy node in cluster.

Will be much easier to use Livy if we can get Livy Server address via some API instead of thinking about how to configure Livy Server and set some configuration manually or by some script during starting Livy.

Using ZK address more easily since a lot of other components typically use ZK and we don't need to perform any additional configuration for our cluster like adding env variables for Livy



address and specifying an additional property in livy.conf. It can provide a better user experience.

The aim here is simplifying configuration. If we already have ZK on our cluster we can configure only one property to use ZK StateStore, Livy Server discovery, Livy ThriftServer discovery, Livy HA etc, and reuse the same ZK cluster configuration (env variables, additional conf files etc) the same way as we do it for other components.

It's not convenient to use ZK as state store with configured ZK address and then add an additional variable for Livy Server address.

When I started to use Livy and found that we can use ZK as state store I tried to find the ability to get Livy Server address as well. I was surprised that this functionality is missed and we should still to specify Livy address manually even though we already use ZK. It will be more easily and conventional to store Livy Server address in ZK as well

Part 2: Technical Specification

Solution. Livy Server discovering via ZooKeeper

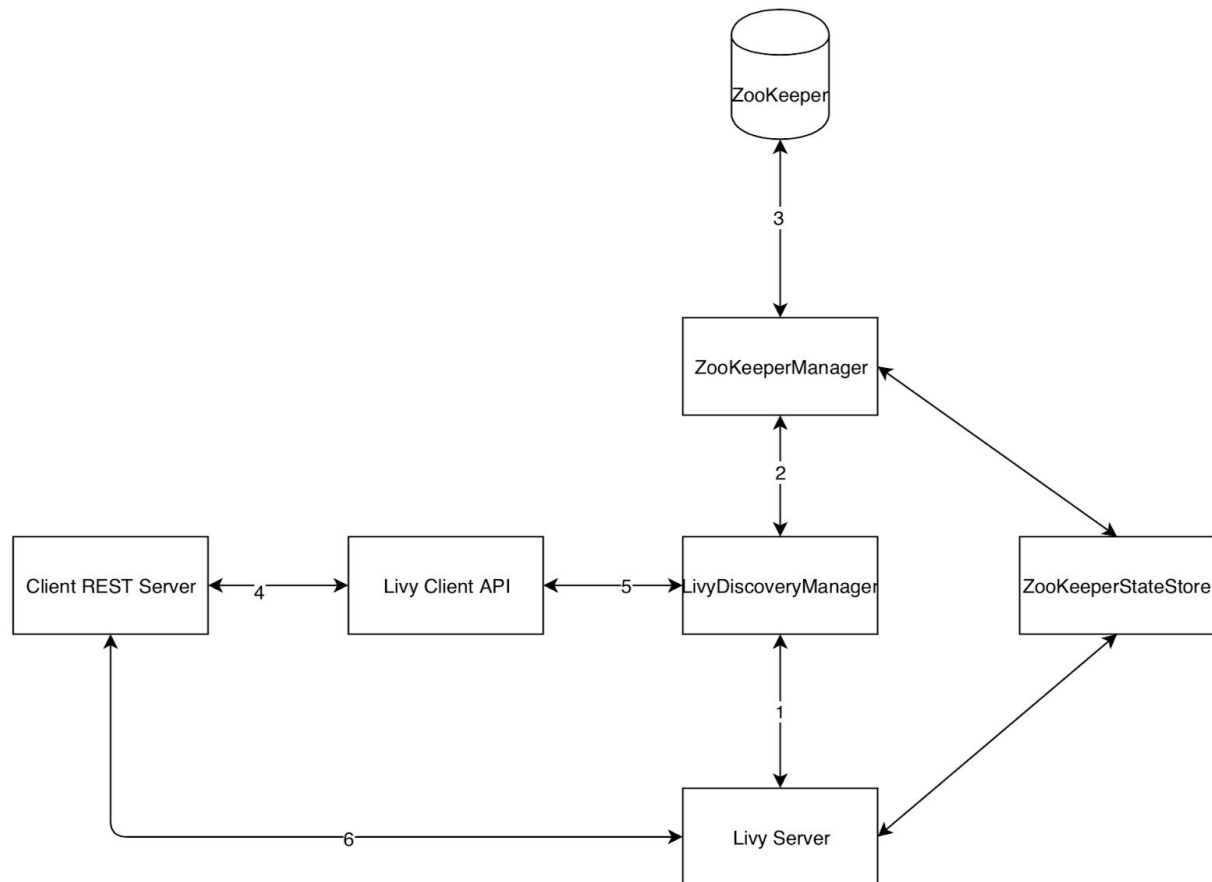
ZooKeeper as backend store for Livy address

Livy already supports storing session information for subsequent recovering in case when we restart Livy. We can store Livy Server URI in ZooKeeper during starting Livy Server. Then we can provide API for clients to get this value and use it in their Java/Scala code.

Implementation

Main idea and architecture

Store Livy Server address in ZooKeeper znode e.g. `/livy/server.uri` during starting Livy Server. We can redesign current solution for ZooKeeper State Store to implement ZooKeeperManager which can be used for implementing LivyDiscoveryManager, ZooKeeperStateStore, ThriftServerDiscovery (I see PR for it) etc.



1. During starting Livy Server call `LivyDiscoveryManager` to save URI.
2. `LivyDiscoveryManager` uses `ZooKeeperManager` to communicate with ZooKeeper quorum.
3. `ZooKeeperManager` save address to ZooKeeper znode ***/livy/server.uri***
4. We use `Livy Client API` to get current Livy Server address
5. `Livy Client` gets address from `LivyDiscoveryManager` which uses ZooKeeper as backend storage.
6. We can use current Livy Server address for communication.

To enable Livy Server discovery we need to set ***livy.zookeeper.url***. During start Livy Server we save URI to ZooKeeper. If we specify host by ***livy.server.host*** then we will use this value, if we use default value "0.0.0.0" the local host name will be stored (`InetAddress.getLocalHost.getHostName`) to be able to communicate from other nodes. We will store address as `java.net.URI` (in format `schema://host:port`, e.g. <http://livyhostname:8998>).



Detailed implementation

Since we already have ZK integration for State Store feature we can reuse existing functionality and extend it by discovery functionality.

1. Move functionality for communication with ZK from ZooKeeperStateStore.scala to separated trait ZooKeeperManager.scala. In ZooKeeperStateStore we will use ZooKeeperManager to store data in ZK. ZooKeeperManager code was not changed except couple small improvement for configurations to make it more flexible (See [Configuration](#) section) and Scala changes like renaming couple methods to fit ZK function names and adding “exist” method to increase readability, also added scaladoc.
2. Move JsonMapper from StateStore.scala to separated class to reuse it in ZooKeeperManager and StateStore
3. Create LivyDiscoveryManager.scala which will use ZooKeeperManager to communicate with ZK to set/get Livy Server address
4. Invoke LivyDiscoveryManager#setServerUri in LivyServer#start method to save address to ZK (if **livy.zookeeper.url** property was specified) (See LivyServer#setServerUri). Take note that if we specify host by **livy.server.host** then we will use this value, if we use default value **0.0.0.0** we need to save the local host name (we get it from **InetAddress.getLocalHost.getHostName**) to be able to communicate from other nodes (See LivyServer#resolvedSeverHost). We will store address as java.net.URI (in format schema://host:port, e.g. <http://livyhostname:8998>).

After these changes we can get Livy Server address directly from LivyDiscoveryManager (See [Client code examples](#)).

Perhaps, we also want to get Livy Server address from Java/Scala Livy clients.

1. Add getServerUri to LivyClient.java interface
Add this method as Future<URI> getServerUri(); since we already have such method in RSCClient.java which extends LivyClient.java and to fit other methods and to honor uniform access principle.
2. Override getServerUri method in HttpClient.java to set URI.
3. Add the same method to LivyScalaClient.scala (See [Client code examples](#))

Configuration

As we are planning to refactor State Store implementation let's see what properties we use for it:

livy.server.recovery.mode	enables recovery
livy.server.recovery.state-store	file system or zookeeper



<code>livy.server.recovery.state-store.url</code>	the path of the state store directory in file system or the address to the Zookeeper servers
---	--

Let's change these configuration to separate ZooKeeper logic from state store logic:

Livy Server discovery configurations

<code>livy.zookeeper.url</code>	ZooKeeper quorum URLs, e.g. host1:port1,host2:port2
<code>livy.zookeeper.namespace</code>	Name of base Livy znode. Default livy (previously was hardcoded and can't be changed)
<code>livy.server.zookeeper.namespace</code>	Name of Livy Server znode. Uses livy.zookeeper.namespace as parent. By default will be /livy/server.uri (previously was hardcoded and can't be changed)
<code>livy.server.zookeeper.connection.max.retries</code>	Number of trials to establish the connection to ZooKeeper quorum (3 by default, previously was hardcoded and can't be changed)
<code>livy.server.zookeeper.connection.retry.interval.ms</code>	Sleep time between connection retries to ZooKeeper quorum (default 500, previously was hardcoded and can't be changed)

ZooKeeper State Store configurations

<code>livy.server.recovery.mode</code>	enables recovery
<code>livy.server.recovery.state-store</code>	file system or zookeeper
<code>livy.server.recovery.state-store.url</code>	the path of the state store directory in file system. For ZooKeeper State Store we need to use <code>livy.zookeeper.url</code> instead

Client code examples

1. Get Livy Server address from `LivyDiscoveryManager`

```
val livyConf = new LivyConf() // needed to get ZooKeeper quorum address
```



```
val discoveryManager = LivyDiscoveryManager(livyConf)
val livyAddress = discoveryManager.getServerUri
HttpRequest(method = HttpMethods.GET, uri = livyAddress)
...
```

2. **Get Livy Server address from LivyClient (if needed)**

```
val livyConf = new LivyConf()
val discoveryManager = LivyDiscoveryManager(livyConf)

val clientBuilder = new LivyClientBuilder()
val livyClient = clientBuilder.setURI(discoveryManager.getServerUri).build()

val scalaClient = new ScalaClient(livyClient)
val livyAddress = scalaClient.getServerUri
```