Arenadata™ Grid

Версия - v2.7

Кластеризация в Arenadata Grid

2020 Arenadata™

Оглавление

1 API кластера **3**
- 1.1 GridCluster . . . 3
- 1.2 ClusterNode . . . 3
- 1.3 Cluster Node Attributes . . . 3
- 1.4 Cluster Node Metrics . . . 4
- 1.5 Local Cluster Node . . . 4

2 Группы кластеров **6**
- 2.1 ClusterGroup . . . 6
- 2.2 Предопределенные группы кластеров . . . 7
- 2.3 Группы кластеров с атрибутами узла . . . 8
- 2.4 Кастомные группы кластеров . . . 9
- 2.5 Объединение кластерных групп . . . 10
- 2.6 Получение узлов из групп кластеров . . . 10
- 2.7 Метрики групп кластеров . . . 10

3 Выбор лидера **12**

4 Конфигурация кластера **14**
- 4.1 TCP/IP Discovery . . . 14
- 4.2 ZooKeeper Discovery . . . 25

5 Нулевое разворачивание **28**
- 5.1 Загрузка однорангового класса . . . 28
- 5.2 Явное разворачивание . . . 29

6 Режимы разворачивания **30**
- 6.1 PRIVATE и ISOLATED . . . 30
- 6.2 SHARED . . . 30
- 6.3 CONTINUOUS . . . 31
- 6.4 Un-Deployment и User версии . . . 31
- 6.5 Конфигурация режимов . . . 31

7 Конфигурация сети **34**

Arenadata Grid обладает расширенными возможностями кластеризации, включая группы логических кластеров и автоматическое обнаружение

Узлы **Arenadata Grid** могут автоматически обнаруживать друг друга, что помогает масштабировать кластер без необходимости перезапуска. Разработчики могут также использовать поддержку гибридного облака **ADG**, позволяющего устанавливать соединение между частными и публичными облаками (Рис.1.).

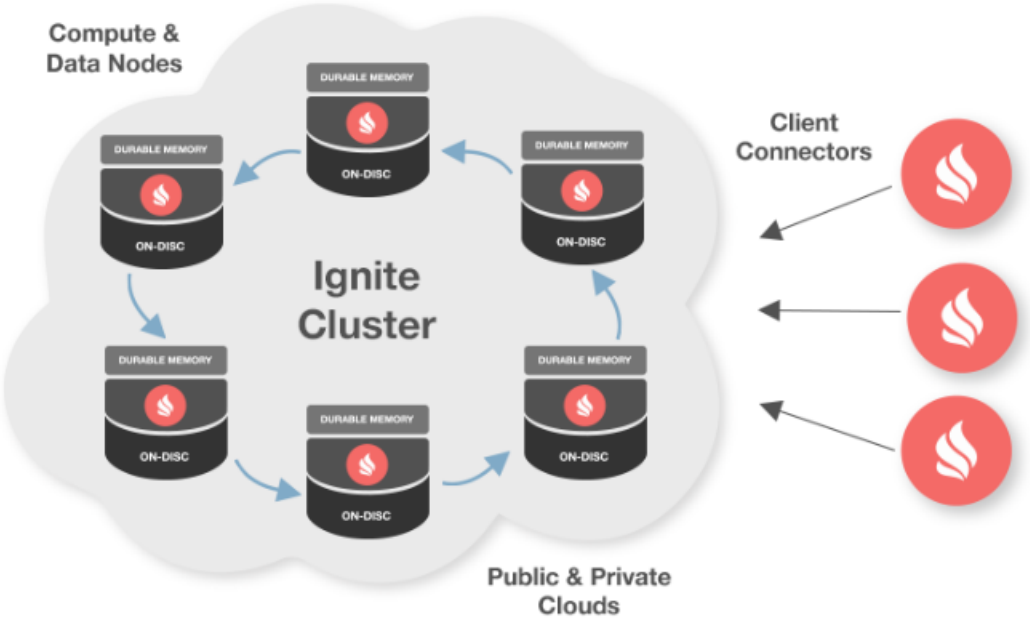


Рис.1.: Кластеризация

В документации приведено полное описание принципа кластеризации в Arenadata Grid – API кластера и его конфигурация, группы кластеров с выбором лидера, возможности разворачивания и настройка сети.

Инструкция может быть полезна администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение системы.

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

Глава 1

API кластера

API для доступа к кластеру и узлам:

- *GridCluster*
- *ClusterNode*
- *Cluster Node Attributes*
- *Cluster Node Metrics*
- *Local Cluster Node*

1.1 GridCluster

Функциональность кластера **Arenadata Grid** обеспечивается через интерфейс **GridCluster**. Экземпляр **GridCluster** из **ADG** можно получить следующим образом:

- Java:

```
Ignite ignite = Ignition.ignite();

IgniteCluster cluster = ignite.cluster();
```

Через интерфейс кластера **ADG** можно выполнить функции:

- Запуск и остановка удаленных узлов кластера;
- Получение списка всех членов кластера;
- Создание логических групп кластеров.

1.2 ClusterNode

Интерфейс **ClusterNode** имеет очень сжатый API и обращается с узлом только в качестве конечной точки логической сети в топологии: его глобальным уникальным идентификатором, метриками узлов, статическими атрибутами, заданными пользователем, и несколькими другими параметрами.

1.3 Cluster Node Attributes

При запуске кластера **Arenadata Grid** все узлы автоматически регистрируют свойства среды и системы как атрибуты узлов. При этом пользователи могут выбрать назначение своих собственных атрибутов узла в

конфигурации:

- XML:

```
<bean class="org.apache.ignite.IgniteConfiguration">
    ...
    <property name="userAttributes">
        <map>
            <entry key="ROLE" value="worker"/>
        </map>
    </property>
    ...
</bean>
```

В следующем примере показано получение узлов с установленным атрибутом “worker”:

- Java:

```
ClusterGroup workers = ignite.cluster().forAttribute("ROLE", "worker");

Collection<ClusterNode> nodes = workers.nodes();
```

Important: Все атрибуты узла доступны с помощью метода *ClusterNode.attribute(“propertyName”)*

1.4 Cluster Node Metrics

Arenadata Grid автоматически собирает метрики для всех узлов в кластере. Метрики собираются в фоновом режиме и обновляются при каждом обмене сообщениями между узлами кластера.

Метрики узлов доступны через интерфейс **ClusterMetrics**, который содержит более *50* различных показателей (для групп кластеров доступны эти же метрики).

Далее приведен пример получения некоторых показателей для локального узла, включая среднюю загрузку процессора и используемую память heap:

- Java:

```
// Local Ignite node.
ClusterNode localNode = cluster.localNode();

// Node metrics.
ClusterMetrics metrics = localNode.metrics();

// Get some metric values.
double cpuLoad = metrics.getCurrentCpuLoad();
long usedHeap = metrics.getHeapMemoryUsed();
int numberOfCores = metrics.getTotalCpus();
int activeJobs = metrics.getCurrentActiveJobs();
```

1.5 Local Cluster Node

Локальный узел кластера – это инстанс **ClusterNode**, представляющий данный узел **ADG**. Пример получения локального узла:

- Java:

```
ClusterNode localNode = ignite.cluster().localNode();
```

Глава 2

Группы кластеров

Логические группы узлов кластера:

- *ClusterGroup*
- *Предопределенные группы кластеров*
- *Группы кластеров с атрибутами узла*
- *Кастомные группы кластеров*
- *Объединение кластерных групп*
- *Получение узлов из групп кластеров*
- *Метрики групп кластеров*

2.1 ClusterGroup

ClusterGroup представляет собой логическую группировку узлов кластера.

В **ADG** все узлы равны по конструкции, поэтому нет необходимости запускать узлы в конкретном порядке или назначать им определенные роли. При этом **Grid** позволяет пользователям логически группировать узлы кластера для любой отдельной цели приложения. Например, можно развернуть сервис только на удаленных узлах или назначить для выполнения задания роль “рабочего” некоторым узлам.

Important: Интерфейс *GridCluster* также является кластерной группой, включающей все узлы в кластере

Выполнение заданий, развертывание сервисов, обмен сообщениями, события и другие задачи можно ограничить для работы только в пределах определенной группы кластеров. В приведенном ниже примере показано как передать задание только на удаленные узлы (исключая локальный узел):

- broadcast:

```
final Ignite ignite = Ignition.ignite();

IgniteCluster cluster = ignite.cluster();

// Get compute instance which will only execute
// over remote nodes, i.e. not this node.
IgniteCompute compute = ignite.compute(cluster.forRemotes());
```

```
// Broadcast to all remote nodes and print the ID of the node
// on which this closure is executing.
compute.broadcast(() -> System.out.println("Hello Node: " + ignite.cluster().localNode().id());
```

- java7 broadcast:

```
final Ignite ignite = Ignition.ignite();

IgniteCluster cluster = ignite.cluster();

// Get compute instance which will only execute
// over remote nodes, i.e. not this node.
IgniteCompute compute = ignite.compute(cluster.forRemotes());

// Broadcast closure only to remote nodes.
compute.broadcast(new IgniteRunnable() {
    @Override public void run() {
        // Print ID of the node on which this runnable is executing.
        System.out.println(">>> Hello Node: " + ignite.cluster().localNode().id());
    }
});
```

2.2 Предопределенные группы кластеров

Группы кластеров можно создавать на основе любого предиката. Для удобства **ADG** поставляется с некоторыми предопределенными группами. Далее приведено несколько примеров групп кластеров, доступных в интерфейсе **ClusterGroup**.

- Remote Nodes:

```
IgniteCluster cluster = ignite.cluster();

// Cluster group with remote nodes, i.e. other than this node.
ClusterGroup remoteGroup = cluster.forRemotes();
```

- Cache Nodes:

```
IgniteCluster cluster = ignite.cluster();

// All nodes on which cache with name "myCache" is deployed,
// either in client or server mode.
ClusterGroup cacheGroup = cluster.forCache("myCache");

// All data nodes responsible for caching data for "myCache".
ClusterGroup dataGroup = cluster.forDataNodes("myCache");

// All client nodes that access "myCache".
ClusterGroup clientGroup = cluster.forClientNodes("myCache");
```

- Nodes With Attributes:

```
IgniteCluster cluster = ignite.cluster();

// All nodes with attribute "ROLE" equal to "worker".
ClusterGroup attrGroup = cluster.forAttribute("ROLE", "worker");
```

- Random Node:

```
IgniteCluster cluster = ignite.cluster();

// Cluster group containing one random node.
ClusterGroup randomGroup = cluster.forRandom();

// First (and only) node in the random group.
ClusterNode randomNode = randomGroup.node();
```

- Host Nodes:

```
IgniteCluster cluster = ignite.cluster();

// Pick random node.
ClusterGroup randomNode = cluster.forRandom();

// All nodes on the same physical host as the random node.
ClusterGroup cacheNodes = cluster.forHost(randomNode);
```

- Oldest Node:

```
IgniteCluster cluster = ignite.cluster();

// Dynamic cluster group representing the oldest cluster node.
// Will automatically shift to the next oldest, if the oldest
// node crashes.
ClusterGroup oldestNode = cluster.forOldest();
```

- Local Node:

```
IgniteCluster cluster = ignite.cluster();

// Cluster group with only this (local) node in it.
ClusterGroup localGroup = cluster.forLocal();

// Local node.
ClusterNode localNode = localGroup.node();
```

- Clients & Servers:

```
IgniteCluster cluster = ignite.cluster();

// All client nodes.
ClusterGroup clientGroup = cluster.forClients();

// All server nodes.
ClusterGroup serverGroup = cluster.forServers();
```

2.3 Группы кластеров с атрибутами узла

Уникальной особенностью **Arenadata Grid** является то, что все узлы равны между собой. Нет master или server узлов, а также нет узлов worker или client. С точки зрения **ADG** они все равны, но при этом пользователи могут настраивать узлы по их усмотрению – как master и server узлы или как worker и client.

Узлы кластера при запуске автоматически регистрируют все свойства среды и системы как атрибуты узлов. Несмотря на это, пользователи могут назначить свои собственные атрибуты узлов с помощью конфигурации **ADG**:

- XML:

```
<bean class="org.apache.ignite.IgniteConfiguration">
    ...
    <property name="userAttributes">
        <map>
            <entry key="ROLE" value="worker"/>
        </map>
    </property>
    ...
</bean>
```

- Java:

```
IgniteConfiguration cfg = new IgniteConfiguration();

Map<String, String> attrs = Collections.singletonMap("ROLE", "worker");

cfg.setUserAttributes(attrs);

// Start Ignite node.
Ignite ignite = Ignition.start(cfg);
```

Important: Атрибуты узла доступны с помощью метода *ClusterNode.attribute("propertyName")*

В следующем примере показано получение узлов с установленным атрибутом "worker":

- Java:

```
IgniteCluster cluster = ignite.cluster();

ClusterGroup workerGroup = cluster.forAttribute("ROLE", "worker");

Collection<GridNode> workerNodes = workerGroup.nodes();
```

2.4 Кастомные группы кластеров

На основе некоторого предиката можно определить динамические кластерные группы. Такие группы всегда включают только узлы, передающие предикат. Ниже приведен пример кластерной группы по узлам, использующим менее *50%* загрузки. При этом узлы в группе со временем меняются в зависимости от нагрузки на них.

- custom group:

```
IgniteCluster cluster = ignite.cluster();

// Nodes with less than 50% CPU load.
ClusterGroup readyNodes = cluster.forPredicate((node) -> node.metrics().getCurrentCpuLoad() < 0.5);
```

- java7 custom group:

```
IgniteCluster cluster = ignite.cluster();

// Nodes with less than 50% CPU load.
ClusterGroup readyNodes = cluster.forPredicate(
    new IgnitePredicate<ClusterNode>() {
```

```
        @Override public boolean apply(ClusterNode node) {
            return node.metrics().getCurrentCpuLoad() < 0.5;
        }
    }
));
```

2.5 Объединение кластерных групп

Группы кластеров можно объединять, вложив их друг в друга. Например, в следующем фрагменте кода показано, как получить рандомный удаленный узел путем объединения удаленной группы с произвольной группой.

- Java:

```
// Group containing oldest node out of remote nodes.
ClusterGroup oldestGroup = cluster.forRemotes().forOldest();

ClusterNode oldestNode = oldestGroup.node();
```

2.6 Получение узлов из групп кластеров

К различным узлам группы кластеров можно перейти следующим образом:

- Java:

```
ClusterGroup remoteGroup = cluster.forRemotes();

// All cluster nodes in the group.
Collection<ClusterNode> grpNodes = remoteGroup.nodes();

// First node in the group (useful for groups with one node).
ClusterNode node = remoteGroup.node();

// And if you know a node ID, get node by ID.
UUID myID = ...;

node = remoteGroup.node(myId);
```

2.7 Метрики групп кластеров

ADG автоматически собирает метрики со всех узлов кластера. Главная особенность в группах кластеров заключается в том, что они автоматически агрегируют метрики по всем узлам в группе и предоставляют соответствующие средние, минимальные и максимальные значения. Метрики доступны через интерфейс **ClusterMetrics**, содержащий более *50* различных показателей (те же самые показатели доступны для отдельных узлов кластера).

Далее приведен пример получения некоторых метрик со всех удаленных узлов, включая среднюю загрузку процессора и используемую память heap:

- Java:

```
// Cluster group with remote nodes, i.e. other than this node.
ClusterGroup remoteGroup = ignite.cluster().forRemotes();
```

```
// Cluster group metrics.
ClusterMetrics metrics = remoteGroup.metrics();

// Get some metric values.
double cpuLoad = metrics.getCurrentCpuLoad();
long usedHeap = metrics.getHeapMemoryUsed();
int numberOfCores = metrics.getTotalCpus();
int activeJobs = metrics.getCurrentActiveJobs();
```

Глава 3

Выбор лидера

При работе в распределенных средах иногда требуется гарантия выбора одного и того же узла, независимо от изменений топологии кластера. Такие узлы называют лидерами.

Во многих системах выбор лидеров кластера обычно связан с согласованностью данных и осуществляется путем сбора голосов от членов кластера. Поскольку в **ADG** согласованность данных обрабатывается функцией индекса соответствия (affinity function, например, Rendezvous Hashing), выбор лидеров в традиционном смысле не требуется.

Тем не менее, все равно может потребоваться узел координатора для определенных задач. С этой целью **ADG** позволяет автоматически выбирать самый старший (oldest node) или самый молодой (youngest node) узел в кластере.

Important: Для большинства вариантов использования leader или singleton рекомендуется функциональность Service Grid, так как она позволяет автоматически разворачивать различные сервисы Cluster Singleton и часто проще в использовании

Самый старший узел обладает свойством оставаться постоянным при добавлении новых узлов. Изменить oldest node возможно только в том случае, когда он удаляется из кластера или аварийно завершает работу. Далее приведен пример выбора группы кластера, содержащей только старший узел:

- Java:

```
IgniteCluster cluster = ignite.cluster();

// Dynamic cluster group representing the oldest cluster node.
// Will automatically shift to the next oldest, if the oldest
// node crashes.
ClusterGroup oldestNode = cluster.forOldest();
```

Самый молодой узел, в отличие от самого старшего, постоянно меняется каждый раз при присоединении новых узлов к кластеру. Иногда это может быть полезно, например, при необходимости выполнения какой-либо задачи на только что присоединенном узле. Далее приведен пример выбора группы кластера, содержащей только youngest node:

- Java:

```
IgniteCluster cluster = ignite.cluster();

// Dynamic cluster group representing the youngest cluster node.
// Will automatically shift to the next youngest, if the youngest
```

```
// node crashes.
ClusterGroup youngestNode = cluster.forYoungest();
```

Глава 4

Конфигурация кластера

Механизм разворачивания **Arenadata Grid** имеет две реализации, предназначенные для разных сценариев использования:

- TCP/IP Discovery, разработанное и оптимизированное для разворачивания *100-300* узлов за *10* секунд;
- *ZooKeeper Discovery*, позволяющий масштабировать кластеры **ADG** от *100* до *1000* узлов, сохраняя линейную масштабируемость и производительность.

4.1 TCP/IP Discovery

В **ADG** узлы могут обнаруживать друг друга с помощью *DiscoverySpi*. Кластер обеспечивает *TcpDiscoverySpi* как реализацию *DiscoverySpi* по умолчанию, использующую TCP/IP для обнаружения узлов. *Discovery SPI* может быть настроен для обнаружения узлов на основе многоадресного поиска (Multicast) и статического IP-поиска (Static IP).

- *Многоадресный IP-поиск*
- *Статический IP-поиск*
- *Одновременное использование Multicast и Static IP*
- *Изолированные кластеры на одном множестве машин*
- *IP-поиск на JDBC*
- *Общая файловая система IP-поиска*
- *ZooKeeper IP-поиск*
- *Время выявления сбоев*
- *Конфигурация TcpDiscoverySpi*

4.1.1 Многоадресный IP-поиск

TcpDiscoveryMulticastIpFinder использует Multicast для обнаружения других узлов в сетке и является IP-поиском по умолчанию. Нет необходимости указывать его, если не планируется переопределение настроек по умолчанию. Далее приведен пример настройки многоадресного IP-поиска:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast.
↪TcpDiscoveryMulticastIpFinder">
          <property name="multicastGroup" value="228.10.10.157"/>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

- Java:

```
TcpDiscoverySpi spi = new TcpDiscoverySpi();

TcpDiscoveryMulticastIpFinder ipFinder = new TcpDiscoveryMulticastIpFinder();

ipFinder.setMulticastGroup("228.10.10.157");

spi.setIpFinder(ipFinder);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default discovery SPI.
cfg.setDiscoverySpi(spi);

// Start Ignite node.
Ignition.start(cfg);
```

4.1.2 Статический IP-поиск

При отключенном Multicast следует использовать *TcpDiscoveryVmIpFinder* с предварительно настроенным списком IP-адресов. Необходимо предоставить хотя бы один IP-адрес удаленного узла, но рекомендуется указать *2* или *3*, даже если их использование планируется в какой-то момент времени в будущем. Как только устанавливается соединение с любым из предоставленных IP-адресов, **ADG** автоматически обнаруживает все остальные узлы сетки.

По умолчанию *TcpDiscoveryVmIpFinder* используется в режиме без общего доступа (non-shared mode). Но если планируется запуск сервер-узла, то в данном режиме список IP-адресов должен содержать адрес локального узла. Это позволяет не ждать присоединения других узлов к кластеру, делает локальный узел первым и запускает нормальную работу. Далее приведен пример настройки статического IP-поиска:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
          <property name="addresses">
            <list>
              <!--
              Explicitly specifying address of a local node to let it start and
↪operate normally even if there is no more nodes in the cluster.
```

```
            You can also optionally specify an individual port or port range.
            -->
            <value>1.2.3.4</value>

            <!--
            IP Address and optional port range of a remote node.
            You can also optionally specify an individual port and don't set                the␣
↪port range at all.
            -->
            <value>1.2.3.5:47500..47509</value>
          </list>
        </property>
      </bean>
    </property>
  </bean>
</property>
</bean>
```

- Java:

```
TcpDiscoverySpi spi = new TcpDiscoverySpi();

TcpDiscoveryVmIpFinder ipFinder = new TcpDiscoveryVmIpFinder();

// Set initial IP addresses.
// Note that you can optionally specify a port or a port range.
ipFinder.setAddresses(Arrays.asList("1.2.3.4", "1.2.3.5:47500..47509"));

spi.setIpFinder(ipFinder);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default discovery SPI.
cfg.setDiscoverySpi(spi);

// Start Ignite node.
Ignition.start(cfg);
```

4.1.3 Одновременное использование Multicast и Static IP

Multicast и Static IP можно использовать одновременно. В этом случае помимо адресов, полученных через многоадресный поиск (если таковые имеются), *TcpDiscoveryMulticastIpFinder* также может работать с предварительно настроенным списком статических IP-адресов. Пример, как настроить многоадресный IP-поиск со статическими IP-адресами:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast.
↪TcpDiscoveryMulticastIpFinder">
          <property name="multicastGroup" value="228.10.10.157"/>

          <!-- list of static IP addresses-->
```

```
          <property name="addresses">
            <list>
              <value>1.2.3.4</value>

              <!--
                  IP Address and optional port range.
                  You can also optionally specify an individual port.
              -->
              <value>1.2.3.5:47500..47509</value>
            </list>
          </property>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

- Java:

```
TcpDiscoverySpi spi = new TcpDiscoverySpi();

TcpDiscoveryMulticastIpFinder ipFinder = new TcpDiscoveryMulticastIpFinder();

// Set Multicast group.
ipFinder.setMulticastGroup("228.10.10.157");

// Set initial IP addresses.
// Note that you can optionally specify a port or a port range.
ipFinder.setAddresses(Arrays.asList("1.2.3.4", "1.2.3.5:47500..47509"));

spi.setIpFinder(ipFinder);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default discovery SPI.
cfg.setDiscoverySpi(spi);

// Start Ignite node.
Ignition.start(cfg);
```

4.1.4 Изолированные кластеры на одном множестве машин

ADG позволяет запускать два изолированных кластера на одном и том же множестве машин. Это возможно, если узлы из разных кластеров используют непересекающиеся локальные диапазоны портов для *TcpDiscoverySpi* и *TcpCommunicationSpi*.

Предположим, что для тестирования необходимо запустить два изолированных кластера на одной машине. Тогда для узлов из первого кластера следует использовать следующие конфигурации *TcpDiscoverySpi* и *TcpCommunicationSpi*:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
    ...
    <!--
    Explicitly configure TCP discovery SPI to provide list of
    initial nodes from the first cluster.
    -->
```

```
    <property name="discoverySpi">
        <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
            <!-- Initial local port to listen to. -->
            <property name="localPort" value="48500"/>

            <!-- Changing local port range. This is an optional action. -->
            <property name="localPortRange" value="20"/>

            <!-- Setting up IP finder for this cluster -->
            <property name="ipFinder">
                <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
                    <property name="addresses">
                        <list>
                            <!--
                            Addresses and port range of nodes from
                            the first cluster.
                            127.0.0.1 can be replaced with actual IP addresses
                            or host names. Port range is optional.
                            -->
                            <value>127.0.0.1:48500..48520</value>
                        </list>
                    </property>
                </bean>
            </property>
        </bean>
    </property>

    <!--
    Explicitly configure TCP communication SPI changing local
    port number for the nodes from the first cluster.
    -->
    <property name="communicationSpi">
        <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
            <property name="localPort" value="48100"/>
        </bean>
    </property>
</bean>
```

- Java:

```
IgniteConfiguration cfg = new IgniteConfiguration();

// Explicitly configure TCP discovery SPI to provide list of initial nodes
// from the first cluster.
TcpDiscoverySpi discoverySpi = new TcpDiscoverySpi();

// Initial local port to listen to.
discoverySpi.setLocalPort(48500);

// Changing local port range. This is an optional action.
discoverySpi.setLocalPortRange(20);

TcpDiscoveryVmIpFinder ipFinder = new TcpDiscoveryVmIpFinder();

// Addresses and port range of the nodes from the first cluster.
// 127.0.0.1 can be replaced with actual IP addresses or host names.
// The port range is optional.
ipFinder.setAddresses(Arrays.asList("127.0.0.1:48500..48520"));
```

```
// Overriding IP finder.
discoverySpi.setIpFinder(ipFinder);

// Explicitly configure TCP communication SPI by changing local port number for
// the nodes from the first cluster.
TcpCommunicationSpi commSpi = new TcpCommunicationSpi();

commSpi.setLocalPort(48100);

// Overriding discovery SPI.
cfg.setDiscoverySpi(discoverySpi);

// Overriding communication SPI.
cfg.setCommunicationSpi(commSpi);

// Starting a node.
Ignition.start(cfg);
```

Для узлов из второго кластера конфигурация может выглядеть следующим образом:

- XML:

```
<bean id="ignite.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">
    <!--
    Explicitly configure TCP discovery SPI to provide list of initial
    nodes from the second cluster.
    -->
    <property name="discoverySpi">
        <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
            <!-- Initial local port to listen to. -->
            <property name="localPort" value="49500"/>

            <!-- Changing local port range. This is an optional action. -->
            <property name="localPortRange" value="20"/>

            <!-- Setting up IP finder for this cluster -->
            <property name="ipFinder">
                <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
                    <property name="addresses">
                        <list>
                            <!--
                            Addresses and port range of the nodes from the second
↪       cluster.
                            127.0.0.1 can be replaced with actual IP addresses or
↪       host names. Port range is optional.
                            -->
                            <value>127.0.0.1:49500..49520</value>
                        </list>
                    </property>
                </bean>
            </property>
        </bean>
    </property>

    <!--
    Explicitly configure TCP communication SPI changing local port number
    for the nodes from the second cluster.
```

```
    -->
    <property name="communicationSpi">
        <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
            <property name="localPort" value="49100"/>
        </bean>
    </property>
</bean>
```

- Java:

```
IgniteConfiguration cfg = new IgniteConfiguration();

// Explicitly configure TCP discovery SPI to provide list of initial nodes
// from the second cluster.
TcpDiscoverySpi discoverySpi = new TcpDiscoverySpi();

// Initial local port to listen to.
discoverySpi.setLocalPort(49500);

// Changing local port range. This is an optional action.
discoverySpi.setLocalPortRange(20);

TcpDiscoveryVmIpFinder ipFinder=new TcpDiscoveryVmIpFinder();

// Addresses and port range of the nodes from the second cluster.
// 127.0.0.1 can be replaced with actual IP addresses or host names.
// The port range is optional.
ipFinder.setAddresses(Arrays.asList("127.0.0.1:49500..49520"));

// Overriding IP finder.
discoverySpi.setIpFinder(ipFinder);

// Explicitly configure TCP communication SPI by changing local port number for
// the nodes from the second cluster.
TcpCommunicationSpi commSpi = new TcpCommunicationSpi();

commSpi.setLocalPort(49100);

// Overriding discovery SPI.
cfg.setDiscoverySpi(discoverySpi);

// Overriding communication SPI.
cfg.setCommunicationSpi(commSpi);

// Starting a node.
Ignition.start(cfg);
```

Как видно из примеров конфигураций, разница незначительна – отличаются только номера портов для SPI и IP поиска.

Для того, чтобы узлы из разных кластеров могли искать друг друга с помощью протокола многоадресного поиска, необходимо заменить *TcpDiscoveryVmIpFinder* на *TcpDiscoveryMulticastIpFinder* и установить уникальный *TcpDiscoveryMulticastIpFinder.multicastGroups* в каждой приведенной выше конфигурации.

Important: Если изолированные кластеры используют персистентность ADG, каждый кластер должен хранить свои файлы персистентности под разными путями в файловой системе. Следует использовать методы *setStoragePath(...)*, *setWalPath(...)* и *setWalArchivePath(...)* в *DataStorageConfiguration* с целью

переопределения путей для каждого отдельного кластера

- Apache jclouds IP Finder
- Amazon S3 IP Finder
- Amazon ELB IP Finder
- Google Cloud Storage IP Finder
- Kubernetes IP Finder

4.1.5 IP-поиск на JDBC

Базу данных можно настроить как общее хранилище исходных IP-адресов, куда узлы записывают свои IP-адреса при запуске. Осуществляется это через *TcpDiscoveryJdbcIpFinder*:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.jdbc.TcpDiscoveryJdbcIpFinder">
          <property name="dataSource" ref="ds"/>
        </bean>
      </property>
    </bean>
  </property>
</bean>

<!-- Configured data source instance. -->
<bean id="ds" class="some.Datasource">
  ...
</bean>
```

- Java:

```
TcpDiscoverySpi spi = new TcpDiscoverySpi();

// Configure your DataSource.
DataSource someDs = MySampleDataSource(...);

TcpDiscoveryJdbcIpFinder ipFinder = new TcpDiscoveryJdbcIpFinder();

ipFinder.setDataSource(someDs);

spi.setIpFinder(ipFinder);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default discovery SPI.
cfg.setDiscoverySpi(spi);

// Start Ignite node.
Ignition.start(cfg);
```

4.1.6 Общая файловая система IP-поиска

Файловую систему можно настроить как общее хранилище для IP-адресов узлов, куда узлы записывают свои IP-адреса при запуске. Осуществляется это через *TcpDiscoverySharedFsIpFinder*:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.sharedfs.TcpDiscoverySharedFsIpFinder
↪">
          <property name="path" value="/var/ignite/addresses"/>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

- Java:

```
// Configuring discovery SPI.
TcpDiscoverySpi spi = new TcpDiscoverySpi();

// Configuring IP finder.
TcpDiscoverySharedFsIpFinder ipFinder = new TcpDiscoverySharedFsIpFinder();

ipFinder.setPath("/var/ignite/addresses");

spi.setIpFinder(ipFinder);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default discovery SPI.
cfg.setDiscoverySpi(spi);

// Start Ignite node.
Ignition.start(cfg);
```

4.1.7 ZooKeeper IP-поиск

При использовании **ZooKeeper** для координации распределенной среды можно применять его как основу общего компонента обнаружения или как IP-поиск. Для настройки **ZooKeeper** в качестве IP-поиска необходимо использовать *TcpDiscoveryZookeeperIpFinder* (при этом модуль ignite-zookeeper должен быть включен):

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
    ...
    <property name="discoverySpi">
        <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
            <property name="ipFinder">
                <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.zk.
↪TcpDiscoveryZookeeperIpFinder">
                    <property name="zkConnectionString" value="127.0.0.1:2181"/>
                </bean>
            </property>
```

```
        </bean>
    </property>
</bean>
```

- Java:

```
TcpDiscoverySpi spi = new TcpDiscoverySpi();

TcpDiscoveryZookeeperIpFinder ipFinder = new TcpDiscoveryZookeeperIpFinder();

// Specify ZooKeeper connection string.
ipFinder.setZkConnectionString("127.0.0.1:2181");

spi.setIpFinder(ipFinder);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default discovery SPI.
cfg.setDiscoverySpi(spi);

// Start Ignite node.
Ignition.start(cfg);
```

4.1.8 Время выявления сбоев

Параметр времени выявления сбоев используется для определения длительности ожидаемого узлом кластера интервала прежде, чем принять удаленное соединение с другим узлом как вышедшее из строя.

Узлы кластера **ADG** связаны друг с другом на уровне обнаружения SPI. Узел отправляет heartbeat-сообщения на второй узел, и если последний не отвечает в установленное время *failDetectionTimeout*, то выводится из кластера.

Данный метод является самым простым способом настройки функции выявления сбоев SPI в зависимости от сетевых и аппаратных условий среды.

Important: *failDetectionTimeout* автоматически контролирует параметры конфигурации *TcpDiscoverySpi*, такие как тайм-аут сокета, тайм-аут подтверждения сообщения и другие. Если какая-либо из данных функций установлена явно, то параметр времени выявления сбоев игнорируется

Время выявления сбоя для узлов сервера **Arenadata Grid** настраивается с помощью *IgniteConfiguration.setFailureDetectionTimeout(long)* и для клиентских узлов – *IgniteConfiguration.setClientFailureDetectionTimeout(long)*. Значение по умолчанию равно *10 секунд* для узлов сервера и *30 секунд* для клиентских узлов. Это гарантирует надежную работу SPI на большинстве локальных и контейнерных разворачиваниях. Однако для стабильных сетей с низкой задержкой параметр может быть установлен примерно на *200 миллисекунд* с целью более быстрого обнаружения сбоев и последующей реакции на них.

4.1.9 Конфигурация TcpDiscoverySpi

Наиболее часто используемые параметры конфигурации *TcpDiscoverySpi* можно увидеть далее в таблице. Полный список параметров конфигурации приведен в *javadoc*.

Таблица4.1.: Параметры конфигурации TcpDiscoverySpi

Метод	Описание	Значение по умолчанию
setIpFinder(TcpDiscoveryIpFinder)	IP-поиск для обмена информацией об IP-адресах узлов	*TcpDiscoveryMulticastIpFinder*. Некоторые реализации для использования: *TcpDiscoverySharedFsIpFinder*, *TcpDiscoveryS3IpFinder*, *TcpDiscoveryJdbcIpFinder*, *TcpDiscoveryVmIpFinder*
setLocalAddress(String)	Устанавка IP-адреса локального хоста, используемого SPI	Если значение не указано, используется первый найденный адрес non-loopback. Если такового нет, используется *java.net.InetAddress.getLocalHost()*
setLocalPort(int)	Прослушивающий SPI порт	*47500*
setLocalPortRange(int)	Диапазон локальных портов. Локальный узел пытается выполнить привязку к первому доступному порту, начиная с локального	*100*
setReconnectCount(int)	Количество попыток узла установить (переустановить) соединение с другим узлом	*2*
setNetworkTimeout(long)	Максимальное время ожидания сети в миллисекундах, используемое для сетевых операций	*5000*
setSocketTimeout(long)	Период ожидания сокета для ограничения времени операций соединения и записи	*2000*
setAckTimeout(long)	Время ожидания для получения подтверждения об отправке сообщения. Если подтверждение не получено в течение установленного периода, отправка считается неудачной, и SPI пытается повторить отправку сообщения	*2000*
setJoinTimeout(long)	Время ожидания соединения. Если используется non-shared IP-поиск и узел не может подключиться к любому адресу с помощью IP-поиска, узел продолжает попытки присоединения в течение установленного периода. Если все адреса по-прежнему не отвечают, возникает исключение, и запуск узла завершается ошибкой. Значение *0* означает бесконечное время ожидания	*0*
setThreadPriority(int)	Приоритет потоков для потоков, запущенных SPI	*0*
setStatisticsPrintFrequency(int)	Статистическая частота печати (в миллисекундах). Значение *0* означает, что печать не требуется. Если значение больше *0*, а журнал не является скрытым, тогда	*true*

4.2 ZooKeeper Discovery

Arenadata Grid использует по умолчанию TCP/IP Discovery, организующую узлы кластера в форме кольцевой топологии, которая имеет свои преимущества и недостатки. Например, в топологиях кластера из сотен узлов системное сообщение может пройти через все узлы за несколько секунд. В результате базовая обработка событий, таких как объединение новых узлов или обнаружение потерпевших сбой, может занять некоторое время, влияя на общую скорость реагирования и производительность кластера.

ZooKeeper Discovery предназначен для массовых разворачиваний **ADG** с сохранением легкости масштабирования и линейной производительности. Однако совместное использование **ADG** и **ZooKeeper** требует настройки и управления двумя распределенными системами, что может оказаться нетривиальной задачей. Поэтому рекомендуется использовать данный SPI метод только в случае, если планируется масштабирование от *100* до *1000* узлов. В противном случае лучше всего использовать TCP/IP Discovery.

ZooKeeper Discovery использует **ZooKeeper** как единую точку синхронизации и организует кластер **ADG** в звездообразную топологию, где кластер **ZooKeeper** находится в центре, и через него обмениваются событиями узлы **ADG** (Рис.4.1.).

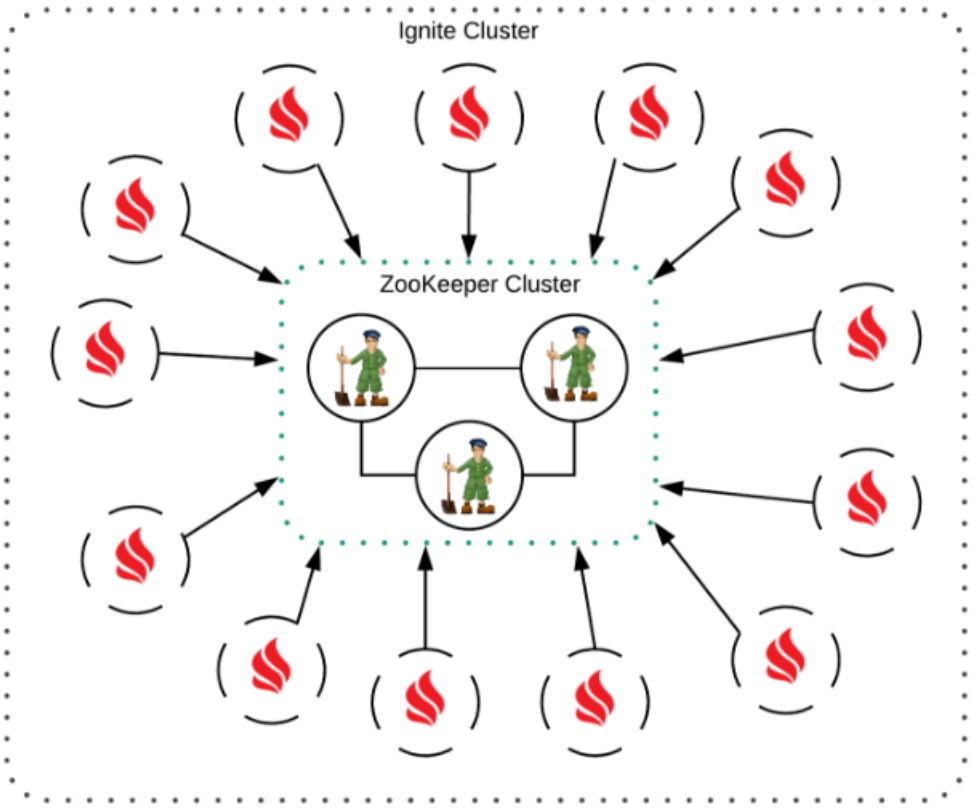


Рис.4.1.: ZooKeeper Discovery

Стоит отметить, что **ZooKeeper Discovery** представляет собой альтернативную реализацию **Grid Discovery SPI**, которая не влияет на **Grid Communication SPI** (см. Конфигурация сети). Как только узлы обнаруживают друг друга через **ZooKeeper Discovery**, они используют **Communication SPI** для одноранговой связи.

- *Конфигурация ZooKeeper Discovery*
- *Неисправности и сплит-контроль*
- *Кастомные события*

- *Анализ конфигурации ADG и ZooKeeper*

4.2.1 Конфигурация ZooKeeper Discovery

Активация **ZooKeeper Discovery** настраивается в *ZookeeperDiscoverySpi* следующим образом:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.zk.ZookeeperDiscoverySpi">
      <property name="zkConnectionString" value="127.0.0.1:34076,127.0.0.1:43310,127.0.0.1:36745"/>
      <property name="sessionTimeout" value="30000"/>
      <property name="zkRootPath" value="/apacheIgnite"/>
      <property name="joinTimeout" value="10000"/>
    </bean>
  </property>
</bean>
```

- Java:

```
ZookeeperDiscoverySpi zkDiscoSpi = new ZookeeperDiscoverySpi();

zkDiscoSpi.setZkConnectionString(
  "127.0.0.1:34076,127.0.0.1:43310,127.0.0.1:36745");
zkDiscoSpi.setSessionTimeout(30_000);

zkDiscoSpi.setZkRootPath("");
zkDiscoSpi.setJoinTimeout(10_000);

IgniteConfiguration cfg = new IgniteConfiguration();

//Override default discovery SPI.
cfg.setDiscoverySpi(zkDiscoSpi);

// Start Ignite node.
Ignition.start(cfg);
```

Обязательно требуются следующие параметры (остальные являются опциональными):

- *zkConnectionString* – сохраняет список адресов серверов ZooKeeper;
- *sessionTimeout* – время, после которого узел ADG считается отключенным, если он не реагирует на обмен событиями через SPI.

4.2.2 Неисправности и сплит-контроль

ZooKeeper Discovery использует метод разделения сети (split brain), сбои связи между отдельными узлами описаны далее.

Предполагается, что **ZooKeeper** всегда отображается для всех узлов кластера. Фактически, если узел отключается от **ZooKeeper**, то данный узел завершает работу, и другие узлы рассматривают его как отказавший или отключенный.

В ситуации разделения сети **ZooKeeper** остается видимым, и узлы не отключаются от него. Однако, так как это случай split brain, некоторые из узлов не могут связаться друг с другом, потому что они расположены в отдельных сегментах сети. В этом случае узлы инициируют процесс устранения сбоев связи, публикуя специальные запросы в кластере **ZooKeeper**. При запуске процесса все узлы пытаются подключиться друг

к другу и отправить результаты узлу-координатору. На основе полученной информации координатор может построить график соединений, найти полностью подключенный компонент и решить, какие узлы отключить, а какие сохранить.

4.2.3 Кастомные события

Изменение кольцевой топологии на звездообразную влияет на способ обработки кастомных событий компонентом **Discovery SPI**. Поскольку кольцевая топология является линейной, это означает, что каждое discovery message обрабатывается узлами последовательно. Таким образом, в любой момент времени сообщение обрабатывает только один узел.

С помощью **ZooKeeper Discovery** координатор отправляет discovery messages всем узлам одновременно, что приводит к параллельной обработке сообщений. В результате такой обработки **ZooKeeper Discovery** не позволяет изменять кастомные события. Например, узлам не разрешено добавлять какую-либо полезную нагрузку в сообщения.

4.2.4 Анализ конфигурации ADG и ZooKeeper

При использовании **ZooKeeper Discovery** требуется уверенность в том, что параметры конфигурации обеих систем соответствуют друг другу и не противоречат. Далее приведен пример конфигурации **ZooKeeper** и последующий анализ.

- Text:

```
# The number of milliseconds of each tick
tickTime=2000

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
```

Настроенный таким образом сервер **ZooKeeper** обнаруживает свою собственную сегментацию из остальной части кластера только после истечения срока “tickTime * syncLimit”. До тех пор, пока это событие не будет обнаружено на уровне **ZooKeeper**, все подключенные к нему узлы **ADG** не производят попытки подключения к другим серверам **ZooKeeper**.

С другой стороны, у **ADG** есть параметр *sessionTimeout*, который определяет, как скоро **ZooKeeper** закрывает сессию узла **ADG** в случае его отключения от кластера **ZooKeeper**. Если данный параметр меньше, чем значение “tickTime * syncLimit”, то узел **ADG** оповещается сегментированным сервером **ZooKeeper** слишком поздно – сессия узла **ADG** истекает прежде, чем он сможет реализовать попытку подключения к другим серверам **ZooKeeper**.

Следовательно, во избежание подобной ситуации значение параметра *sessionTimeout* должно быть больше, чем “tickTime * syncLimit”.

Глава 5

Нулевое разворачивание

Замыкания и задачи, используемые для вычислений, могут иметь любой кастомный класс, включая анонимный. В **ADG** удаленные узлы автоматически узнают об этих классах, и нет необходимости явно разворачивать или перемещать *.jar*-файлы на удаленные узлы.

- *Загрузка одноранговой класса*
- *Явное разворачивание*

5.1 Загрузка одноранговой класса

Использование нулевого разворачивания Zero Deployment возможно благодаря загрузке одноранговых классов (загрузка класса P2P) и специальному распределенному механизму **ClassLoader** в **ADG** для обмена байт-кодом между узлами. При включенной одноранговой загрузке нет необходимости вручную разворачивать код **Java** или **Scala** на каждом узле сетки и повторно разворачивать при каждом его изменении.

Подобный приведенному ниже пример кода будет выполняться на всех удаленных узлах благодаря загрузке одноранговой класса без какого-либо явного шага разворачивания:

- Java:

```
IgniteCluster cluster = ignite.cluster();

// Compute instance over remote nodes.
IgniteCompute compute = ignite.compute(cluster.forRemotes());

// Print hello message on all remote nodes.
compute.broadcast(() -> System.out.println("Hello node: " + cluster.localNode().id()));
```

Загрузка одноранговой класса может быть настроена следующим образом:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
    ...
    <!-- Explicitly enable peer class loading. -->
    <property name="peerClassLoadingEnabled" value="true"/>
    ...
</bean>
```

- Java:

```
IgniteConfiguration cfg = new IgniteConfiguration();

cfg.setPeerClassLoadingEnabled(true);

// Start Ignite node.
Ignite ignite = Ignition.start(cfg);
```

Последовательность загрузки однорангового класса работает по принципу:

1. **ADG** проверяет, доступен ли класс по локальному пути *classpath* (т.е. был ли он загружен при запуске системы). Если класс загружен, он возвращается. В этом случае загрузка класса из однорангового узла не состоится.
2. Если класс локально недоступен, тогда на исходный узел отправляется запрос на предоставление определения класса. Исходный узел отправляет определение байт-кода класса, и класс загружается на рабочем узле. Это происходит только один раз для каждого класса – после определения класса на узле его больше не придется загружать снова.

Important: При использовании загрузки однорангового класса необходимо знать библиотеки, которые загружаются из одноранговых узлов и библиотек, которые уже доступны локально в пути к классу. Рекомендуется включить все сторонние библиотеки в путь класса каждого узла. Это можно сделать, скопировав файлы *JAR* в папку *Grid libs*. Таким образом, каждый раз при изменении строки кода мегабайты сторонних классов не будут передаваться на удаленные узлы

5.2 Явное разворачивание

Для явного разворачивания *JAR*-файлов в **ADG** необходимо скопировать их в папку *libs* на каждом члене кластера, и они будут автоматически загружены при запуске **ADG**.

Глава 6

Режимы разворачивания

Специфика поведения по принципу одноранговой загрузки контролируется различными режимами разворачивания. В частности, в случаях, когда исходный или мастер-узел покидает сетку, поведение зависит от режима разворачивания. Другими аспектами, управляемыми режимом разворачивания, являются управление пользовательскими ресурсами и версиями классов.

- *PRIVATE и ISOLATED*
- *SHARED*
- *CONTINUOUS*
- *Un-Deployment и User версии*
- *Конфигурация режимов*

6.1 PRIVATE и ISOLATED

Классы, развернутые в одном ClassLoader на мастер-узле, совместно используют один и тот же загрузчик классов на рабочих узлах. Однако задачи, развернутые из разных мастер-узлов, не используют один и тот же ClassLoader на рабочих узлах, что полезно при разработке – разработчики могут работать с разными версиями одних и тех же классов.

Различий в режимах PRIVATE и ISOLATED не существует. Обе константы сохранены по причинам обратной совместимости, и одна из них, вероятно, будет удалена в будущем.

Если мастер-узел покидает кластер, то в данном режиме классы не разворачиваются.

6.2 SHARED

Режим разворачивания по умолчанию. В режиме SHARED классы из разных мастер-узлов с одной и той же пользовательской версией совместно используют один и тот же ClassLoader на рабочих узлах. Классы отменяются при каждом выходе всех мастер-узлов из сетки или при изменении пользовательской версии. Данный режим позволяет классам, исходящим из разных мастер-узлов, совместно использовать одни и те же экземпляры пользовательских ресурсов на удаленных узлах. Этот метод особенно полезен в рабочей среде, поскольку по сравнению с режимом ISOLATED, который имеет область одного загрузчика класса на одном мастер-узле, режим SHARED расширяет область разворачивания до всех мастер-узлов.

Если все мастер-узлы покидают кластер, то в данном режиме классы не разворачиваются.

6.3 CONTINUOUS

В режиме CONTINUOUS классы не отменяются даже в случае, когда мастер-узлы покидают сетку. Отмена разворачивания происходит только при изменении пользовательской версии класса. Преимущество данного подхода заключается в том, что он позволяет выполнять задачи, исходящие из разных мастер-узлов, которые совместно используют одни и те же инстансы пользовательских ресурсов на рабочих узлах. Это позволяет выполнять все задачи на рабочих узлах повторно, например, одни и те же инстансы пулов соединений или кэшей. При использовании режима CONTINUOUS можно запустить несколько автономных рабочих узлов, определить пользовательские ресурсы на мастер-узлах и инициализировать их один раз на рабочих узлах независимо от того, с какого мастер-узла они развернуты. По сравнению с режимом ISOLATED, который имеет область одного загрузчика класса на одном мастер-узле, режим CONTINUOUS расширяет область разворачивания на все мастер-узлы, что особенно полезно в рабочей среде.

Даже если все мастер-узлы покидают кластер, то в данном режиме классы не отменяются.

6.4 Un-Deployment и User версии

Определения классов, полученные при одноранговой загрузке, имеют собственный жизненный цикл. В некоторых случаях (при выходе мастер-узла или изменении пользовательской версии – в зависимости от режима разворачивания), информация о классе не разворачивается из кластера: класс стирается со всех узлов в сетке, и пользовательские ресурсы, связанные с данным классом определения, также могут быть удалены (в зависимости от режима разворачивания).

Пользовательская версия вступает в игру всякий раз при повторном разворачивании классов в режимах SHARED или CONTINUOUS. По умолчанию **ADG** автоматически определяет, изменился ли загрузчик классов или перезапущен узел. При необходимости изменения и повторном разворачивании кода в подмножестве узлов в случае режима CONTINUOUS, следует убить каждое живое разворачивание и изменить пользовательскую версию.

Пользовательская версия указывается в файле *META-INF/ignite.xml* пути класса следующим образом:

- XML:

```
<!-- User version. -->
<bean id="userVersion" class="java.lang.String">
    <constructor-arg value="0"/>
</bean>
```

По умолчанию все скрипты запуска **ADG** (*ignite.sh* или *ignite.bat*) поднимают пользовательскую версию из папки *IGNITE_HOME/config/userversion*. Как правило, ее достаточно просто обновить в данной папке. Однако в случае разворачивания *GAR* или *JAR* следует запомнить файл *META-INF/ignite.xml* с требуемой пользовательской версией.

6.5 Конфигурация режимов

Следующие параметры конфигурации для одноранговой загрузки классов могут быть дополнительно настроены в *IgniteConfiguration*:

Таблица6.1.: Конфигурация в IgniteConfiguration

Метод	Описание	Значение по умолчанию
setPeerClassLoadingEnabled(boolean)	Включение/отключение одноранговой загрузки классов	*false*
setPeerClassLoadingExecutorService (ExecutorService)	Настройка пула потоков одноранговой загрузки классов. Если не установлено, используется значение по умолчанию	*null*
setPeerClassLoadingExecutorService Shutdown(boolean)	Флаг выключения сервиса одноранговой загрузки класса. При значении *true* пул потоков загрузки однорангового класса принудительно отключается в случае остановки узла	*true*
setPeerClassLoadingLocalClassPath Exclude(String...)	Список пакетов в пути системного класса, который должен быть загружен P2P, даже если они находятся локально	*null*
setPeerClassLoadingMissedResources CacheSize(int)	Размер кэша пропущенных ресурсов. Значение *0* устанавливается во избежание кэширования пропущенных ресурсов	*100*
setDeploymentMode(DeploymentMode)	Режим разворачивания для задач и классов	*SHARED*

Пример конфигурации одноранговой загрузки классов:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
    <!--
        Explicitly enable peer class loading. Set to false
        to disable the feature.
    -->
    <property name="peerClassLoadingEnabled" value="true"/>

    <!-- Set deployment mode. -->
    <property name="deploymentMode" value="CONTINUOUS"/>

    <!-- Disable missed resources caching. -->
    <property name="peerClassLoadingMissedResourcesCacheSize" value="0"/>

    <!--
        Exclude force peer class loading of a class,
        even if exists locally.
    -->
    <property name="peerClassLoadingLocalClassPathExclude">
        <list>
            <value>com.mycompany.MyChangingClass</value>
        </list>
    </property>
```

```
</bean>
```

- Java:

```
IgniteConfiguration cfg=new IgniteConfiguration();

// Explicitly enable peer class loading.
cfg.setPeerClassLoadingEnabled(true);

// Set deployment mode.
cfg.setDeploymentMode(DeploymentMode.CONTINUOUS);

// Disable missed resource caching.
cfg.setPeerClassLoadingMissedResourcesCacheSize(0);

// Exclude force peer class loading of a class,
// even if it exists locally.
cfg.setPeerClassLoadingLocalClassPathExclude("com.mcompany.MyChangingClass");

// Start a node.
Ignition.start(cfg);
```

Глава 7

Конфигурация сети

CommunicationSpi обеспечивает базовый канал для отправки и получения сообщений сетки и используется для всех распределенных сетевых операций, таких как выполнение задач, мониторинг обмена данными, запрос распределенных событий и другие. **ADG** предоставляет *TcpCommunicationSpi* в качестве реализации *CommunicationSpi* по умолчанию, которая использует TCP/IP для связи с другими узлами.

Для включения связи с другими узлами *TcpCommunicationSpi* добавляет атрибуты локального узла *TcpCommunicationSpi.ATTR_ADDRS* и *TcpCommunicationSpi.ATTR_PORT*. При запуске данный SPI пытается начать прослушивание локального порта, указанного методом *TcpCommunicationSpi.setLocalPort(int)*. Если локальный порт занят, то SPI автоматически увеличивает номер порта до тех пор, пока не сможет успешно подключиться для прослушивания. Параметр конфигурации *TcpCommunicationSpi.setLocalPortRange(int)* управляет максимальным количеством портов, к которым SPI пытается подключиться.

Important: Диапазон портов очень удобен при запуске нескольких узлов сетки на одном компьютере или даже на одной виртуальной машине. В этом случае все узлы могут быть подняты без каких-либо изменений конфигурации

Следующие параметры конфигурации сети могут быть дополнительно настроены в *TcpCommunicationSpi*:

Таблица7.1.: Конфигурация в TcpCommunicationSpi

Метод	Описание	Значение по умолчанию
setLocalAddress(String)	Адрес локального хоста для привязки сокетов	Любой доступный локальный IP-адрес
setLocalPort(int)	Локальный порт для привязки сокетов	*47100*
setLocalPortRange(int)	Максимальное количество локальных портов, если все ранее проверенные порты заняты	*100*
setTcpNoDelay(boolean)	Значение параметра сокета *TCP_NODELAY*. Каждый принятый или созданный сокет использует указанное значение. Для сокращения времени на операцию запрос/ответ при передаче по протоколу TCP параметр должн быть установлен на значение *true* (по умолчанию). В большинстве случаев не рекомендуется изменять данный параметр	*true*
setConnectTimeout(long)	Время ожидания подключения при установлении соединения с удаленными узлами	*1000*
setIdleConnectionTimeout(long)	Максимальное время простоя соединения, по истечении которого соединение с клиентом закрывается	*30000*
setBufferSizeRatio(double)	Коэффициент размера буфера для SPI. По мере отправки сообщений размер буфера регулируется с помощью данного отношения	*0.8* или *IGNITE_COMMUNICATION_BUF_RESIZE_RATIO*, если установлено
setMinimumBufferedMessage Count(int)	Минимальное количество сообщений для SPI, которые буферизуются перед отправкой	*512* или *IGNITE_MIN_BUFFERED_COMMUNICATION_MSG_CNT*, если установлено
setDualSocketConnection (boolean)	Флаг о принудительном соединении пары сокетов между узлами. Если задано значение *true*, между взаимодействующими узлами устанавливается два отдельных соединения: одно для исходящих сообщений и одно для входящих. При значении *false* для обоих направлений используется одно TCP-соединение. Опция полезна в некоторых операционных системах, когда флаг *TCP_NODELAY* отключен, а доставка сообщений занимает слишком много времени	*false*
setConnectionBufferSize(int)	Параметр используется, если для *setAsyncSend(boolean)*	*0*

Пример конфигурации сети:

- XML:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  ...
  <property name="communicationSpi">
    <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
      <!-- Override local port. -->
      <property name="localPort" value="4321"/>
    </bean>
  </property>
  ...
</bean>
```

- Java:

```
TcpCommunicationSpi commSpi = new TcpCommunicationSpi();

// Override local port.
commSpi.setLocalPort(4321);

IgniteConfiguration cfg = new IgniteConfiguration();

// Override default communication SPI.
cfg.setCommunicationSpi(commSpi);

// Start grid.
Ignition.start(cfg);
```