

---

# **Amino.Run Documentation**

*Release RC7*

**The Amino.Run Developers**

**May 23, 2019**



<b>1</b>	<b>What is Amino.Run?</b>	<b>3</b>
<b>2</b>	<b>Why we created Amino.Run?</b>	<b>5</b>
<b>3</b>	<b>Documentation</b>	<b>7</b>
<b>4</b>	<b>References</b>	<b>9</b>
<b>5</b>	<b>Amino.Run Overview</b>	<b>11</b>
5.1	Microservices . . . . .	11
5.2	OMS . . . . .	12
5.3	Kernel Server . . . . .	12
5.4	DM . . . . .	12
5.5	Kernel Object . . . . .	12
<b>6</b>	<b>Remote Interfaces</b>	<b>13</b>
6.1	KernelServer . . . . .	13
<b>7</b>	<b>RMI Registry</b>	<b>15</b>
<b>8</b>	<b>gRPC vs RMI</b>	<b>17</b>
<b>9</b>	<b>Kernel Server &amp; OMS</b>	<b>19</b>
<b>10</b>	<b>Stubs</b>	<b>21</b>
10.1	App_Stub . . . . .	21
10.2	ClientPolicy . . . . .	22
10.3	ServerPolicy_Stub . . . . .	23
10.4	KernelClient . . . . .	23
10.5	KernelServer . . . . .	24
<b>11</b>	<b>Introduction</b>	<b>27</b>
11.1	KeepInCloud + LockingTransactions + ConsensusRSM . . . . .	27
<b>12</b>	<b>Enabling Client to reach the Server behind a NAT</b>	<b>29</b>
12.1	Problem description: . . . . .	29
12.2	Proposed solution . . . . .	29
12.3	Steps to implement the proposed solution . . . . .	30

<b>13 Quick Start</b>	<b>33</b>
13.1 Download and install GraalVM Community Edition . . . . .	33
13.2 Install Android SDK and Android Studio (optional) . . . . .	33
13.3 Get and Build the Source Code . . . . .	34
<b>14 Communicate</b>	<b>37</b>
<b>15 Some additional Background Reading for the Curious</b>	<b>39</b>
15.1 Releasing . . . . .	39
<b>16 Amino.Run Documentation</b>	<b>41</b>
<b>17 Create a new document</b>	<b>43</b>
<b>18 Contribute to existing documentation</b>	<b>45</b>

Amino.Run is an open source, multilanguage development platform and distributed runtime environment designed to make distributed applications much easier to design, develop and operate.



---

## What is Amino.Run?

---

Amino.Run is an open source, multilanguage development platform and distributed runtime environment designed to make distributed applications much easier to design, develop and operate. It is the distributed process runtime component of the broader Amino distributed operating system effort, which also includes subsystems for:

1. distributed, reactive memory (Amino.Sync)
2. distributed, transactional persistent store (Amino.Store)
3. distributed privacy and security (Amino.Safe)

Amino.Run supports most commonly used programming languages and is, by default, deployed and managed in container environments using [Kubernetes](#). Common examples of distributed applications include:

1. [mobile-cloud applications](#),
2. [mobile backend style applications](#),
3. [edge computing applications](#) and
4. [other cloud-native applications](#)

All of these classes of applications share a common set of difficult design and development challenges including performance, distributed concurrency, remote invocation, synchronization, fault tolerance, scalability, sharding, code and data migration, leader election, load balancing, observability, fault diagnosis and many more.

Amino.Run is based on, and extends, several years of research work done at the [University Of Washington Computer Systems Lab](#) in Seattle.<sup>1,2,3</sup>

Parts of Amino.Run are based on the [Sapphire](#) open source project related to the above research, and which is used in compliance with [Sapphire's MIT License](#).

Amino.Run is alpha software, and not yet suitable for production use. We have a well-funded development team actively working on getting it production ready, and actively support contributions from the open source community.





---

### Why we created Amino.Run?

---

In a nutshell, we created Amino.Run to make design, development and operation of reliable, fast, distributed applications quicker, easier and more fun.

In summary, our approach is to:

1. provide a wide and expandable range of standard, re-usable, pluggable and production-ready Deployment Managers **DMs** to solve many common distributed computing problems (including all of those mentioned above) so that you can focus on application logic, not solving hard distributed systems challenges.
2. make it very easy to plug combinations of these into new or existing application code, even if it was not designed to be distributed - in many cases a few lines of code can change a simple standalone application written to run on a single computer into a robust, scalable distributed, cloud-native application. Create sharded, consistent replicas of your objects, or replicated shards. Either way it requires only a one-line code or configuration change to your application.
3. support a wide variety of programming languages - we recognise the need for different languages and embrace that need. Java, Javascript, Python, C++, Swift, Ruby, Rust and others<sup>4</sup>- we've got you covered - and without the need for clunky and inefficient REST or RPC library code to get them to talk to each other. We encourage using multiple different languages to develop different parts of a single application.
4. make it easy to deploy your application anywhere, and move it around (piece by piece) as you wish:
  1. On your local machine
  2. On (public or private) cloud servers
  3. On mobile devices (Android, iOS)
  4. On edge devices
  5. Even have Amino.Run move parts of your application around automatically at runtime to optimize performance, reliability or battery power consumption.

Over time Amino will include a runtime process manager (Amino.Run), a reactive distributed memory manager (Amino.Sync), a consistent transactional storage system (Amino.Sync) and a privacy and security framework (Amino.Safe). Initial focus is on making Amino.Run production-ready.



## CHAPTER 3

---

### Documentation

---

- *Amino.Run Overview*
- *Multi DM*
- *Getting started*
- *Amino.Run Examples*
- **Contributing:**
  - *Setting up your developer environment*
  - *Contributing to the documentation*
- *External documentation web site*



## CHAPTER 4

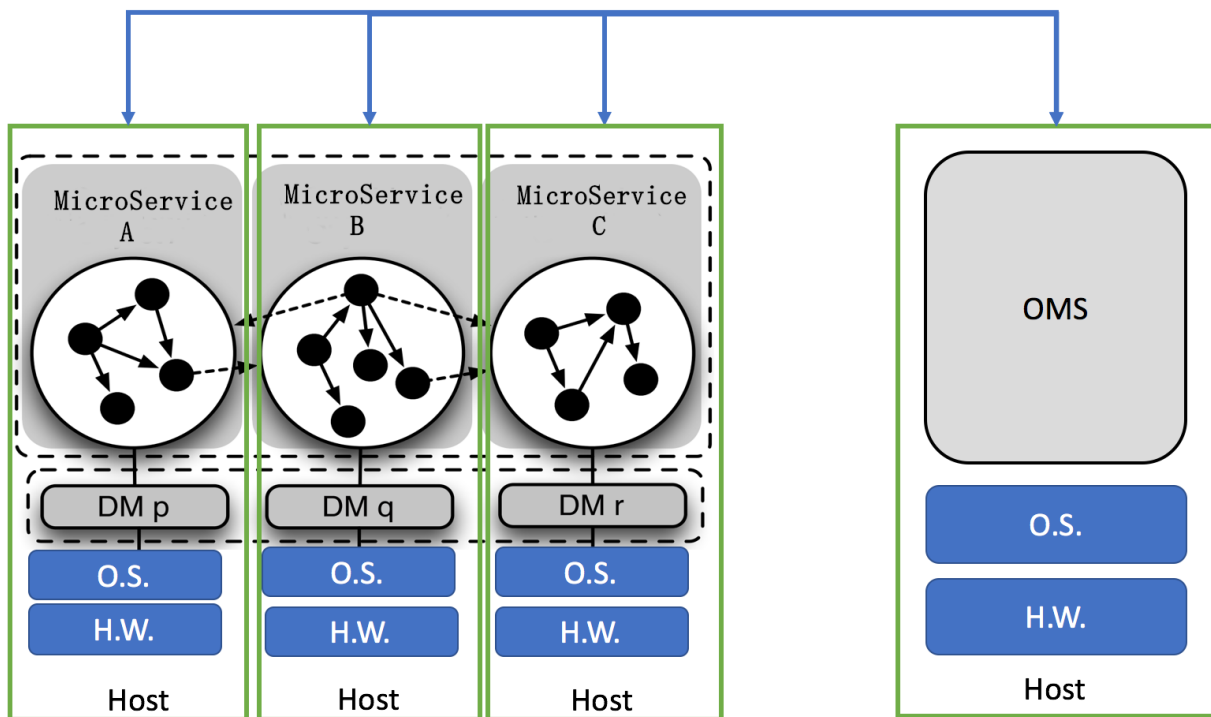
---

### References

---

1 Customizable and Extensible Deployment for Mobile/Cloud Applications 2 Diamond: Automating Data Management and Storage for Wide-area, Reactive Applications 3 Building Consistent Transactions with Inconsistent Replication (Extended Version) 4 Not all of these languages are currently officially supported, but they are all on our medium-term roadmap, support based on GraalVM





## 5.1 Microservices

Microservices are the base management unit in Amino. In the above diagram, each circle represents one Microservice. The dots inside the circle (i.e. the Microservice) represent normal programming language (e.g. Java) objects. One Microservice may contain a set of such objects. The solid arrow lines between dots are *local* method invocations between objects. The dashed arrow lines between circles are *remote* method invocations between Microservices.

Methods on normal Java objects can only be invoked *locally* by objects residing on the same host. Microservices may however have *remote methods* which can be invoked by objects residing on different hosts.

Deployment Kernel has the capability to move a Microservice from one host to another. Behind the scenes, Deployment Kernel will serialize the whole Microservice, including all programming language objects belonging to the Microservice, on one end, ship the data to the destination host, and then do the deserialization there.

Microservices are created by applications using a static helper method `Microservice.new_()`. To invoke a method on a Microservice, applications have to first get a reference to the Microservice from OMS Server.

## 5.2 OMS

OMS, Object Management Service, keeps track of the location of all Microservices. Unlike normal (e.g. Java) objects which can be created using the Java `new` keyword, microservices must be created with a special Amino.Run helper method 'Microservice.new\_()'. Upon Microservice creation, the method `MicroService.new_()` will generate a globally unique ID for the Microservice, and register the object in OMS. OMS provides API to search Amino Run. Given a Microservice ID, OMS can tell the IP of the host on which the Microservice runs. Whenever a Microservice is moved or deleted, OMS will be updated accordingly.

## 5.3 Kernel Server

Kernel Server provides runtime environment for Amino Run. Each host runs a Kernel Server instance. Kernel Server exposes a set of *remote* API which can be invoked remotely. Amino.Run assumes that any Kernel Server can invoke the *remote* API on any other Kernel Server regardless where the Kernel Server lives.

## 5.4 DM

Every DM, Deployment Manager, has three components: a proxy, a instance manager, and a coordinator. When users create Microservice, he/she can optionally associate a DM to the Microservice. Not all Microservice has DMs. But if a DM is specified for a Microservice, then during the creation of the Microservice, helper method `MicroService.new_()` will inject code into the `stub` of the Microservice, in which case any method invocation on the Microservice will first be processed by the `proxy`, `instance manager` and the `coordinator` of the DM before reach the actual Microservice. Each DM provides one specific functionality. The Sapphire paper listed 26 DMs.

## 5.5 Kernel Object

*Kernel object* is a wrapper of the *actual* (e.g. Java) object - it contains a reference to the actual object and exposes a `invoke` method which allows any public methods defined on the actual object to be invoked with reflection.

Kernel objects are created with `KernelServerImpl.newKernelObject` method. Every kernel object has a unique `oid` and is registered in OMS server. `KernelServer` interface also exposes a few APIs to copy and move kernel objects.



---

## Remote Interfaces

---

Amino.Run declares two Remote interfaces: `KernelServer` and `OMSServer`. Most methods in these two interfaces can be easily replaced with gRPC, except for `KernelServer.copyKernelObject`.

A note about code snippets: The code below is out-of-date. Many of the class and package names no longer apply, but the general principles do.

### 6.1 KernelServer

```
public interface KernelServer extends Remote {
    Object makeKernelRPC(KernelRPC rpc) throws RemoteException,
↳KernelObjectNotFoundException, KernelObjectMigratingException, KernelRPCException;
    void copyKernelObject(KernelOID oid, sapphire.kernel.server.KernelObject
↳object) throws RemoteException, KernelObjectNotFoundException;
    AppObjectStub startApp(String className) throws RemoteException;
}
```

###OMSServer

```
public interface OMSServer extends Remote {
    KernelOID registerKernelObject(InetSocketAddress host) throws RemoteException;
    void registerKernelObject(KernelOID oid, InetSocketAddress host) throws
↳RemoteException, KernelObjectNotFoundException;
    InetSocketAddress lookupKernelObject(KernelOID oid) throws RemoteException,
↳KernelObjectNotFoundException;

    ArrayList<InetSocketAddress> getServers() throws NumberFormatException,
↳RemoteException, NotBoundException;
    ArrayList<String> getRegions() throws RemoteException;
    InetSocketAddress getServerInRegion(String region) throws RemoteException;

    void registerKernelServer(InetSocketAddress host) throws RemoteException,
↳NotBoundException;
```

(continues on next page)

(continued from previous page)

```
MicroserviceID registerMicroservice(EventHandler dispatcher) throws ↳
↳RemoteException;
ReplicaID registerReplica(MicroserviceID oid, EventHandler dispatcher) throws ↳
↳RemoteException, MicroserviceNotFoundException;
EventHandler getMicroserviceDispatcher(MicroserviceID oid) throws ↳
↳RemoteException, MicroserviceNotFoundException;
EventHandler getReplicaDispatcher(ReplicaID rid) throws RemoteException, ↳
↳MicroserviceNotFoundException;

/* Called by the client */
public AppObjectStub getAppEntryPoint() throws RemoteException;
}
```

As a remote procedure call framework, gPRC does not provide a mechanism to move *objects* from one host to another. *Objects* are different from gRPC *messages* because objects may have methods. But we can build this *object moving* capability on top of gRPC by taking the following three actions:

- Serialize the object into a byte stream on client side
- Pass the byte stream to server side by calling a gRPC function on the server
- Deserialize byte stream into object on server side

However there is one catch. In order to deserialize an object, the server on which the object will be deserialized needs the access to the class definition. `ClassNotFoundException` will be thrown if the server cannot find the class definition on the class path. Unlike RMI, gRPC is not able to dynamically download class definition from a remote location, therefore we need to build up a mechanism to allow servers to download jar files remotely.

In the first phase, to keep things simple, we can assume that there is one single jar file which contains all class definitions and this jar file is accessible to all servers.

## CHAPTER 7

---

### RMI Registry

---

Amino.Run uses RMI registry to discover remote objects. The following snippet shows how to register remote object *io.amino.run.oms*.

```
// Register io.amino.run.oms
OMSServerImpl oms = new OMSServerImpl(args[2]);
sapphire.oms.OMSServer omsStub = (sapphire.oms.OMSServer) UnicastRemoteObject.
    ↪exportObject(oms, 0);
Registry registry = LocateRegistry.createRegistry(port);
registry.rebind("io.amino.run.oms", omsStub);
```

Client side lookups remote object OMSServer by its name, i.e. *io.amino.run.oms*, and RMI registry returns a *stub* of OMSServer. Client then uses OMS server to look up Amino Run.

```
// Look up io.amino.run.oms
registry = LocateRegistry.getRegistry(args[0], Integer.parseInt(args[1]));
OMSServer server = (OMSServer) registry.lookup("io.amino.run.oms");

// Look up Microservice from OMS Server
TwitterManager tm = (TwitterManager) server.getAppEntryPoint();
System.out.println("Received Twitter Manager Stub: " + tm);
```

If we switch to gRPC, we can no longer use RMI registry. We need to come up with another mechanism to register and lookup gRPC servers. It should be straightforward to do.



## CHAPTER 8

---

### gRPC **VS** RMI

---

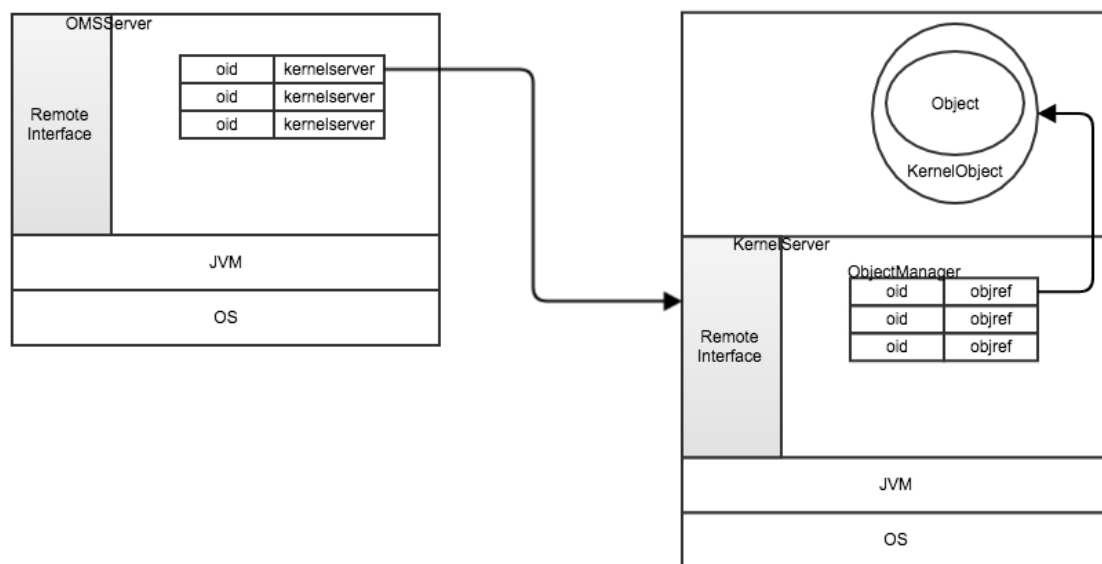
Amino.Run uses RMI in its internal implementation. Applications running on top of Amino.Run do not have to use RMI. Replacing RMI with gRPC, if done properly, should have little impact on Amino.Run applications. But application developers must make one change: Amino.Run applications can no longer use RMI registry to find OMS server location; they have to switch to gRPC service discovery mechanism.



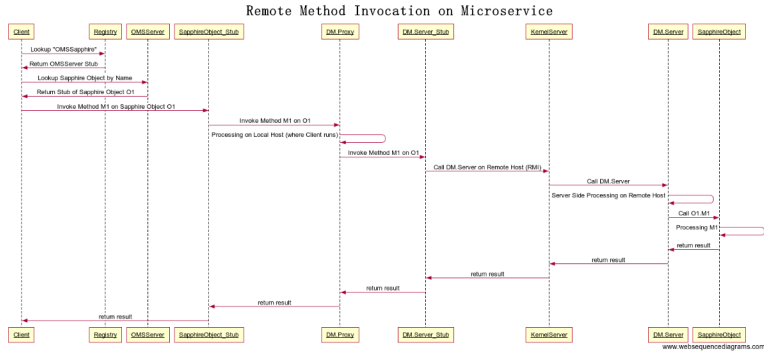
## Kernel Server & OMS

KernelServer and OMSServer are two important objects in Amino.Run. Both expose remote interfaces. OMSServer contains a KernelObjectManager which keeps track of the mapping between kernel object ID to the IP address of the kernel server in which the object runs. Given an kernel object ID, a client can call OMSServer to get the IP of the host where the object runs.

KernelServer contains a ObjectManager which keeps track of the mapping between kernel object ID to the reference of the object.

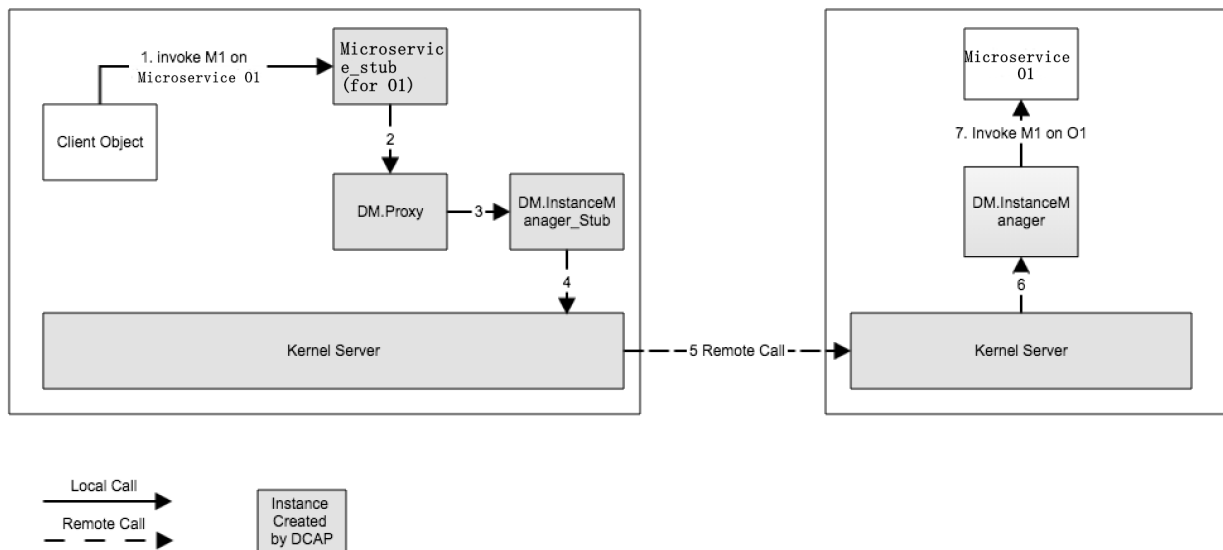


The following sequence chart demonstrate the high level interactions between client, OMS Server, and Kernel Server.





Amino.Run generates many `stub` classes. The following chart shows the relationship between these `stub` classes. We then use the source code to explain how these `stubs` work together to process a remote method invocation.



## 10.1 App\_Stub

`App_Stub` contains `$_client` which is a reference to `ClientPolicy`. Every method call on `App` object will be translated to an `onRPC` call on embedded `$_client`.

```

public final class UserManager_Stub extends sapphire.appexamples.minnietwitter.app.
↳UserManager implements sapphire.common.AppObjectStub {

    // holds a reference to client policy
  
```

(continues on next page)

(continued from previous page)

```

        amino.run.policy.Policy.Client $__client = null;

        // Implementation of addUser(String, String)
        public sapphire.appexamples.minnietwitter.app.User addUser(java.lang.String
        ↪$param_String_1, java.lang.String $param_String_2)
            throws sapphire.app.AppObjectNotCreatedException {
            java.lang.Object $__result = null;
            try {
                if ($__directInvocation)
                    $__result = super.addUser($param_String_1, $param_String_2);
                else {
                    ↪java.util.ArrayList<Object> $__params = new java.util.ArrayList
        ↪<Object>();
                    String $__method = "public sapphire.appexamples.minnietwitter.app.
        ↪User sapphire.appexamples.minnietwitter.app.UserManager.addUser(java.lang.String,
        ↪java.lang.String) throws sapphire.app.AppObjectNotCreatedException";
                    $__params.add($param_String_1);
                    $__params.add($param_String_2);
                    $__result = $__client.onRPC($__method, $__params);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return ((sapphire.appexamples.minnietwitter.app.User) $__result);
        }

```

## 10.2 ClientPolicy

The onRPC call on ClientPolicy will be translated to an onRPC call on ServerPolicy\_Stub.

```

public abstract class DefaultPolicyUpcallImpl extends amino.run.policy.PolicyLibrary {

    public abstract static class DefaultClientUpcallImpl extends amino.run.policy.
    ↪PolicyLibrary.ClientLibrary {
        public Object onRPC(String method, ArrayList<Object> params) throws ↪
    ↪Exception {
            // The default client behavior is to just perform the RPC
            // to the Policy Server
            Object ret = null;
            try {
                ret = getServer().onRPC(method, params);
            } catch (RemoteException e) {
                setServer(getGroup().onRefRequest());
            }
            return ret;
        }
    }

    public abstract static class DefaultServerPolicyUpcallImpl extends ↪
    ↪ServerPolicyLibrary {
        public Object onRPC(String method, ArrayList<Object> params) throws ↪
    ↪Exception {
            // The default server behavior is to just invoke
            // the method on the Microservice this Server

```

(continues on next page)

(continued from previous page)

```

        // Policy Object manages
        return appObject.invoke(method, params);
    }
}

```

## 10.3 ServerPolicy\_Stub

ServerPolicy\_Stub uses the embedded KernelClient to do a makeKernelRPC call. It tries to use makeKernelRPC call to invoke DefaultServerPolicyUpcallImpl.onRPC method on the remote kernel server.

```

public final class CacheLeasePolicy$CacheLeaseServerPolicy_Stub extends amino.run.
↳policy.cache.CacheLeasePolicy.CacheLeaseServerPolicy implements sapphire.kernel.
↳common.KernelObjectStub {
    // Implementation of onRPC(String, ArrayList)
    public java.lang.Object onRPC(java.lang.String $param_String_1, java.util.
↳ArrayList $param_ArrayList_2)
        throws java.lang.Exception {
        java.util.ArrayList<Object> $__params = new java.util.ArrayList<Object>();
        String $__method = "public java.lang.Object amino.run.policy.
↳DefaultPolicyUpcallImpl$DefaultServerPolicyUpcallImpl.onRPC(java.lang.String, java.
↳util.ArrayList<java.lang.Object>) throws java.lang.Exception";
        $__params.add($param_String_1);
        $__params.add($param_ArrayList_2);
        java.lang.Object $__result = null;
        try {
            $__result = $__makeKernelRPC($__method, $__params);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return $__result;
    }

    public Object $__makeKernelRPC(java.lang.String method, java.util.ArrayList
↳<Object> params) throws java.rmi.RemoteException, java.lang.Exception {
        sapphire.kernel.common.KernelRPC rpc = new sapphire.kernel.common.KernelRPC($
↳_oid, method, params);
        try {
            return sapphire.kernel.common.GlobalKernelReferences.nodeServer.
↳getKernelClient().makeKernelRPC(this, rpc);
        } catch (sapphire.kernel.common.KernelObjectNotFoundException e) {
            throw new java.rmi.RemoteException();
        }
    }
}

```

## 10.4 KernelClient

KernelClient makes a RMI call on a remote KernelServer with server.makeKernelRPC(rpc).

```

public class KernelClient {
    public Object makeKernelRPC(KernelObjectStub stub, KernelRPC rpc) throws
↳KernelObjectNotFoundException, Exception {
        InetAddress host = stub.$__getHostname();
        logger.info("Making RPC to " + host.toString() + " RPC: " + rpc.
↳toString());

        // Check whether this object is local.
        KernelServer server;
        if (host.equals(GlobalKernelReferences.nodeServer.getLocalHost())) {
            server = GlobalKernelReferences.nodeServer;
        } else {
            server = getServer(host);
        }

        // Call the server
        try {
            return tryMakeKernelRPC(server, rpc);
        } catch (KernelObjectNotFoundException e) {
            return lookupAndTryMakeKernelRPC(stub, rpc);
        }
    }

    private Object tryMakeKernelRPC(KernelServer server, KernelRPC rpc) throws
↳KernelObjectNotFoundException, Exception {
        Object ret = null;
        try {
            ret = server.makeKernelRPC(rpc);
        } catch (KernelRPCException e) {
            throw e.getException();
        } catch (KernelObjectMigratingException e) {
            Thread.sleep(100);
            throw new KernelObjectNotFoundException("Kernel object was
↳migrating. Try again later.");
        }
        return ret;
    }
}

```

## 10.5 KernelServer

The remote KernelServer receives the makeKernelRPC call. It locates the object in objectManager and then calls the method on the object.

```

@Override
    public Object makeKernelRPC(KernelRPC rpc) throws RemoteException,
↳KernelObjectNotFoundException, KernelObjectMigratingException, KernelRPCException {
        sapphire.kernel.server.KernelObject object = null;
        object = objectManager.lookupObject(rpc.getOID());

        logger.info("Invoking RPC on Kernel Object with OID: " + rpc.getOID()
↳+ " with rpc:" + rpc.getMethod() + " params: " + rpc.getParams().toString());
        Object ret = null;
        try {
            ret = object.invoke(rpc.getMethod(), rpc.getParams());

```

(continues on next page)

(continued from previous page)

```
        } catch (Exception e) {
            throw new KernelRPCException(e);
        }
        return ret;
    }
    ...
}
```

#### # AppEntry Creation

Every application written using Amino.Run has one `AppEntryPoint` which is the starting point of the application. The following sequence chart shows how a client (e.g. `TwitterWorldGenerator`) gets the `AppEntryPoint` (e.g. `MinnieTwitterStart`) from OMS, and how OMS creates `AppEntryPoint` on Kernel Server behind the scene.





Multiple DM's may be associated with each MicroService. This document describes some combinations of DM's, and how these combinations behave and might be useful.

In general, DM's in the same category are mutually exclusive, and it does not usually make sense to combine them. For example, KeepInCloud and KeepOnDevice do not make sense together - choose one or the other. Similarly, choose between LockingTransactions and OptimisticTransactions, and not both.

Conversely, by combining DM's in different categories, new and often very useful deployment behaviors can be achieved without having to write any code. For example, by combining ConsensusRSM, KeepInCloud, and LockingTransactions, it is possible to get the union of their behaviors, namely:

1. A RAFT-based replicated state machine (ConsensusRSM).
2. All replicas remain in a given cloud zone (KeepInCloud).
3. Multi-operation read-write transactions using server-side locking (LockingTransactions)

## 11.1 KeepInCloud + LockingTransactions + ConsensusRSM

### 11.1.1 Desired behavior

1. Client creates a new instance or obtains a reference to an existing MicroService.
2. Client starts a locking transaction, by calling startTransaction()
3. Client invokes multiple read and write operations against the MicroService.
4. Client either commits or rolls back the transaction.
5. Client expects the MicroService to be highly available, resilient to server machine failures (provided that concurrent failures are limited to a minority quorum).
6. Client expects the MicroService to be high performance (all quorum communication is on the local zone network).
7. Client does not expect the MicroService to be resilient to zone failure.

### 11.1.2 How it works under the hood

1. Client creates an instance of a `MicroService` (`_new()`).
2. Kernel invokes `group.onCreate()` on all DM's (some handwaving here, but I think we can make it work).
  1. `KeepInCloud.group.onCreate()` ensures that all replicas are in the required cloud zone.
  2. `LockingTransactions.group.onCreate()` does nothing unusual.
  3. `ConsensusRSM.group.onCreate()` creates  $2f+1$  replicas (by invoking `sapphire_replicate`, which in turn invokes `addServer` on all DM's).
3. Client starts a locking transaction, by calling `startTransaction()` on the `MicroService`
4. The above is intercepted by `KeepInCloud.client.onRPC()`, that does nothing other than `server.onRPC()`.
5. The above is intercepted by `LockingTransactions.client.onRPC()` that identifies `startTransaction()` and acquires a server-side lock by invoking `LockingTransactions.server.acquireLock()`.
6. The above is intercepted by `ConsensusRSM.client.onRPC()`, that invokes the RAFT consensus algorithm across all replicas to ensure that the RPC call (`acquireLock()` in this case) is committed against the quorum. (note that in the current implementation, `LockingTransactions.server` issues the lease identifier (a random UUID). Given that there will be  $2f+1$  servers in this case,  $2f+1$  different lease id's would be issued. So to make this work, there should be an option to have the lease ID generated on the client - `LockingTransactions.client` - to ensure that the lock identifier is consistent across all replicas. This change should be straightforward.



---

## Enabling Client to reach the Server behind a NAT

---

### 12.1 Problem description:

When the server, S1, lives behind a NAT, the client, C1, cannot reach it, because

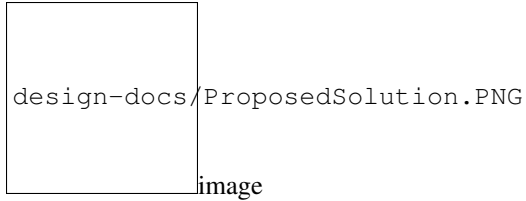
- C1 cannot use the External IP address: when S1 intends to reach the world outside the NAT, the NAT dynamically translates S1's internal IP address to an external IP address. The external IP address is dynamic, i.e., only valid for the duration of the session, hence not reachable by C1 in the future
- Internal IP address not reachable by C1: the internal IP address is behind the NAT and hence packets destined to this IP address get blocked by the NAT



### 12.2 Proposed solution

We make use of a proxy server, with a static external IP address, in the middle. Here is how it works:

1. The server, S1, establishes a VPN connection, as a VPN client, to the VPN server running on the proxy server
2. The VPN server in the proxy server allocates an IP address to S1 and maintains the mapping between S1 and this IP address (VPN-assigned IP address)
3. An nginx proxy running on the proxy server that allocates a port on the proxy server node to S1 (nginx-assigned port). The nginx proxy maintains the mapping between (Proxy\_Server\_IP:Ngix\_Assigned\_port, S1\_VPN\_Assigned\_IP:S1\_Port). It forwards any incoming request from C1 to S1 based on this mapping



## 12.3 Steps to implement the proposed solution

### 1. Set up Proxy Server

- Any node with a public IP address can be used
- E.g., we created an Ubuntu 16.04 VM on AWS with the public IPv4 address 34.208.50.35
- Note that UDP port 1194 should be open from firewall.

### 2. Install OpenVPN Server on Proxy Server

- Run the following script. Make sure you input the public IP address during the setup

```
$ wget https://git.io/vpn -O openvpn-install.sh && bash openvpn-install.sh
```

- Note that the IP address of this server will be “10.8.0.1” by default. Therefore, connection from other client should be to “10.8.0.1”, not the public IP address
- When creating a client profile, name it different from the default value “client”. For example, “client1” should work. It will generate “client1.ovpn” which will be later used in client set up

### 3. Install Nginx Proxy on Proxy Server

- On Proxy Server run:

```
$ sudo apt-get install nginx  
$ sudo systemctl start nginx
```

### 4. Install OpenVPN Client on S1

- For Ubuntu run:

```
$ sudo apt-get install openvpn
```

- For Android install Android OpenVPN Connect on Google Play Store

### 5. Connect S1 to OpenVPN Server

- For Ubuntu: use the configuration created in Step 2 to connect to the OpenVPN Server

```
$ sudo openvpn --config client1.ovpn
```

- Log message should display the assigned IP address (e.g., “Connected: SUCCESS, 10.8.0.3,18.219.220.105,1194” 10.8.0.3 is assigned client IP address)

### 6. Update nginx proxy to add forwarding rules

- Edit nginx config under /etc/nginx/sites-enabled/default to add forwarding rules for S1. Remove the existing setting inside location bracket (e.g., “try\_files ...”) and replace with the below ‘proxy\_pass’ example (IP address should match to the new client).

```
server {  
    listen 22345 default_server;  
    listen [::]:22345 default_server;  
    ...  
    location / {  
        proxy_pass http://10.8.0.3:22345;  
    }  
}
```

- Restart nginx

```
$ sudo systemctl restart nginx
```



## 13.1 Download and install GraalVM Community Edition

- You will need to download and install the correct version (usually the latest stable version) based on the dependency configured in `core/build.gradle`. As of April 2019, that's [GraalVM Community Edition 1.0 RC8](#). Note that the open source Community Edition works fine, so don't bother with the Enterprise Edition unless you have a specific need for it. Follow instructions at <https://www.graalvm.org/docs/getting-started/> for downloading and installing. In particular, set your `JAVA_HOME` and `PATH` variables appropriately. For example, something along the lines of the following at the end of your `~/.bash_profile` in your home directory works well on Linux and Mac OS X:

```
export GRAALVM_HOME=~Downloads/graalvm-ce-1.0.0-rc8/Contents/Home
export JAVA_HOME=$GRAALVM_HOME
export PATH=$GRAALVM_HOME/bin:$PATH
```

After then, you need to install ruby support for GraalVM:

```
gu install ruby
```

## 13.2 Install Android SDK and Android Studio (optional)

- Android SDK and Android Studio are *not* required by Amino. But many Amino demo applications are android applications. We recommend installing Android SDK and Android Studio.
- Follow [instructions](#) to install Android SDK and Android Studio. More details can be found at [here](#).

```
// on Mac
$ brew cask install android-sdk
$ brew cask install android-ndk
```

### 13.2.1 Accept Android SDK License

```
// on Mac
$ /usr/local/share/android-sdk/tools/bin/sdkmanager --licenses
```

### 13.2.2 Add Android Properties

```
> cd Amino.Run/
> cat >> local.properties << EOF
ndk.dir=<your ndk dir>
sdk.dir=<your sdk dir>
EOF
```

## 13.3 Get and Build the Source Code

### 13.3.1 Check out from Github

```
# checkout Amino.Run
$ git clone https://github.com/amino-os/Amino.Run
> cd Amino.Run/core
```

### 13.3.2 Build and Test the Core

```
> ../gradlew build
```

### 13.3.3 Build and Run Basic Example Applications

```
> ./gradlew :examples:run
```

### 13.3.4 Other Gradle Tasks and Tips

#### List Projects

```
> ./gradlew projects
```

#### List All Gradle Tasks

```
> ./gradlew tasks --all
```

#### Clean All Build Artifacts

```
> ./gradlew clean
```

### Format Source Code

```
> ./gradlew goJF  
# verify source code style  
> ./gradlew verGJF
```

### Generate Policy Stub

```
> cd Amino.Run/core  
> ../gradlew genStubs
```

### 13.3.5 Other Gradle Tips

```
> ./gradlew properties  
> ./gradlew jar
```





## CHAPTER 14

---

### Communicate

---

- To join our [Slack](#) channels, send your public GitHub account, Slack account name and email address to Sungwook Moon (sungwook.moon@huawei.com)



---

## Some additional Background Reading for the Curious

---

- Read Papers
- Read Sapphire source code
- Read Code Study Notes
- Review the Principles of Distributed Systems. If you have not done a university course on distributed systems you will need to read the following to get a basic understanding of the principles and common terminology. Feel free to add more resource links below.
  - UMass Course 677
  - Distributed System Principles by Andrew Tanenbaum

### 15.1 Releasing

#### 15.1.1 Publish Core to Bintray

```
export BINTRAY_USER="<bintray\_user>"
export BINTRAY_API_KEY="<bintray\_api\_key>"

> ./gradlew --info :core:bintrayUpload

# publish apache harmony to bintray
> ./gradlew --info :dependencies:apache.harmony:bintrayUpload
```



## CHAPTER 16

---

### Amino.Run Documentation

---

Documentation is hosted on [Read the Docs](#) We welcome community contributions to expand and improve it.



---

## Create a new document

---

Please follow the steps below to add new documents

- All the documents are written in [GitHub's markdown style](#)
- You will need to create a markdown file and place the file in the /docs path of the repository
- If you have any images then please place them in the /docs/images path of the repository
- Next identify where you want to provide the link to your document on the [Amino Documentation](#) web-page
- To add the link you will have to edit the index.rst file. Please be very cautious while editing this files as this file defines the layout and navigation for [Amino.Run Documentation](#) web-page. [How it works?](#)
- Once you are satisfied with the changes you want to make please commit you changes and raise a [Pull Request](#)
- Once your PR is merged post review you will see the changes on the [Amino.Run Documentation](#) web-page





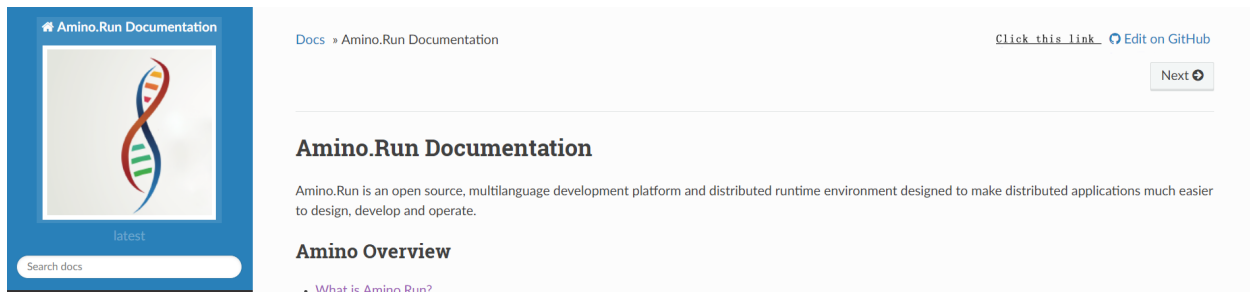
---

## Contribute to existing documentation

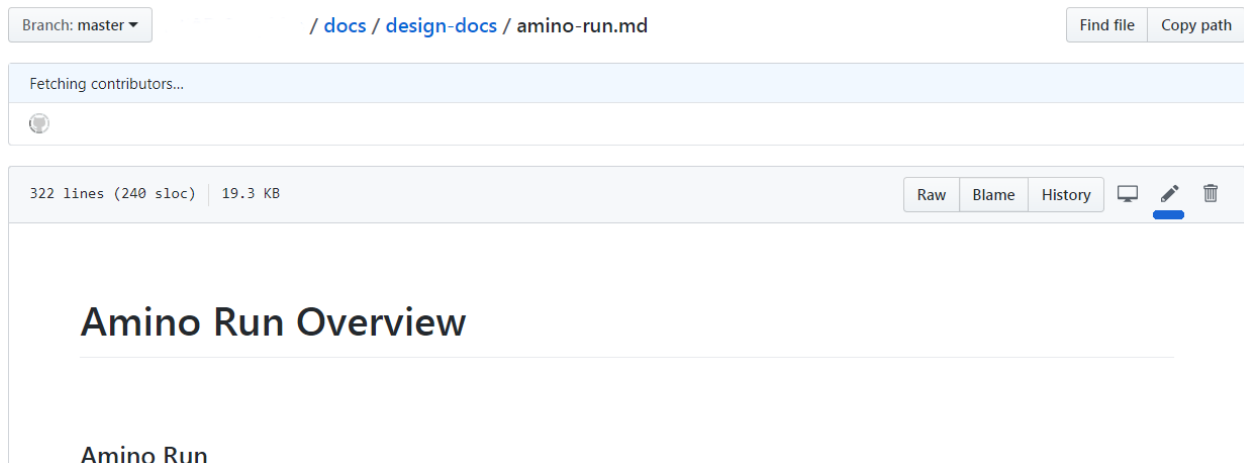
---

If you want to help in improving any of the document then please follow the below guide lines

- All the documents are written in [GitHub's markdown style](#)
- Every document web-page has an “Edit On GitHub” link at the top right hand side corner of the screen
- Click this link and you will be redirected to the GitHub page where this document resides



- Once you are on the GitHub page you can click on the Edit icon and start modifying the document



- If you have any images then please place them in the /docs/images path of the repository
- Once you are satisfied with the changes you want to make please commit your changes and raise a [Pull Request](#)
- Once your PR is merged post review you will see the changes on the [Amino Documentation](#) web-page