Akka Cluster with Spring BOOT

Spring BOOT AkkaCluster , Spring Boot Application .

,

. AKKACluster( )

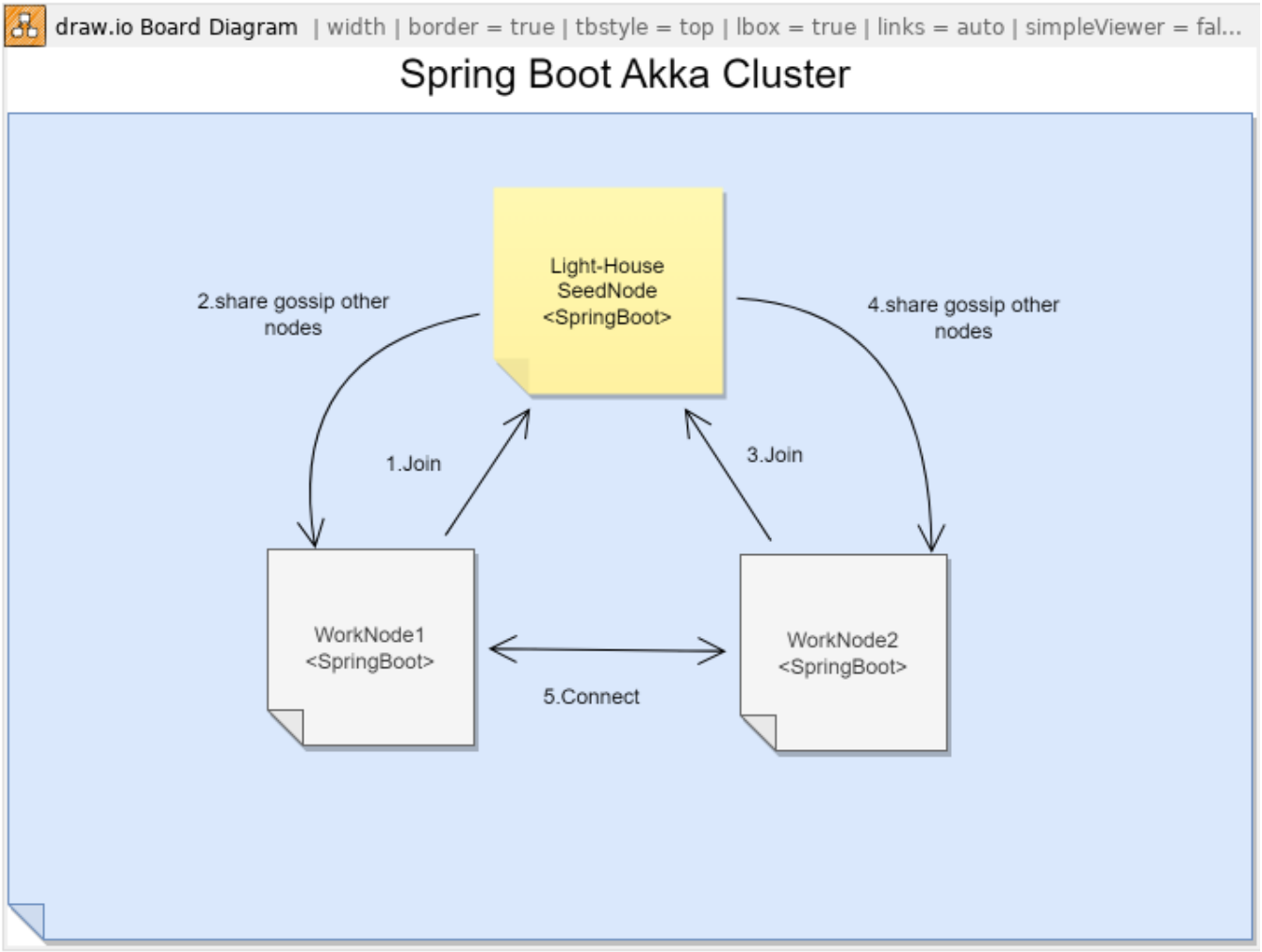
Spring Boot .

- : https://github.com/psmon/java-labs

	• Rest TCP(netty) .	• Rest .
TPS	• Throttle / TPS .	• Kafka TPS .
Role	• .	• .
	• .	• .
BackPresure	• Role .	• .

AkkaCluster Flow .

Cluster gossip

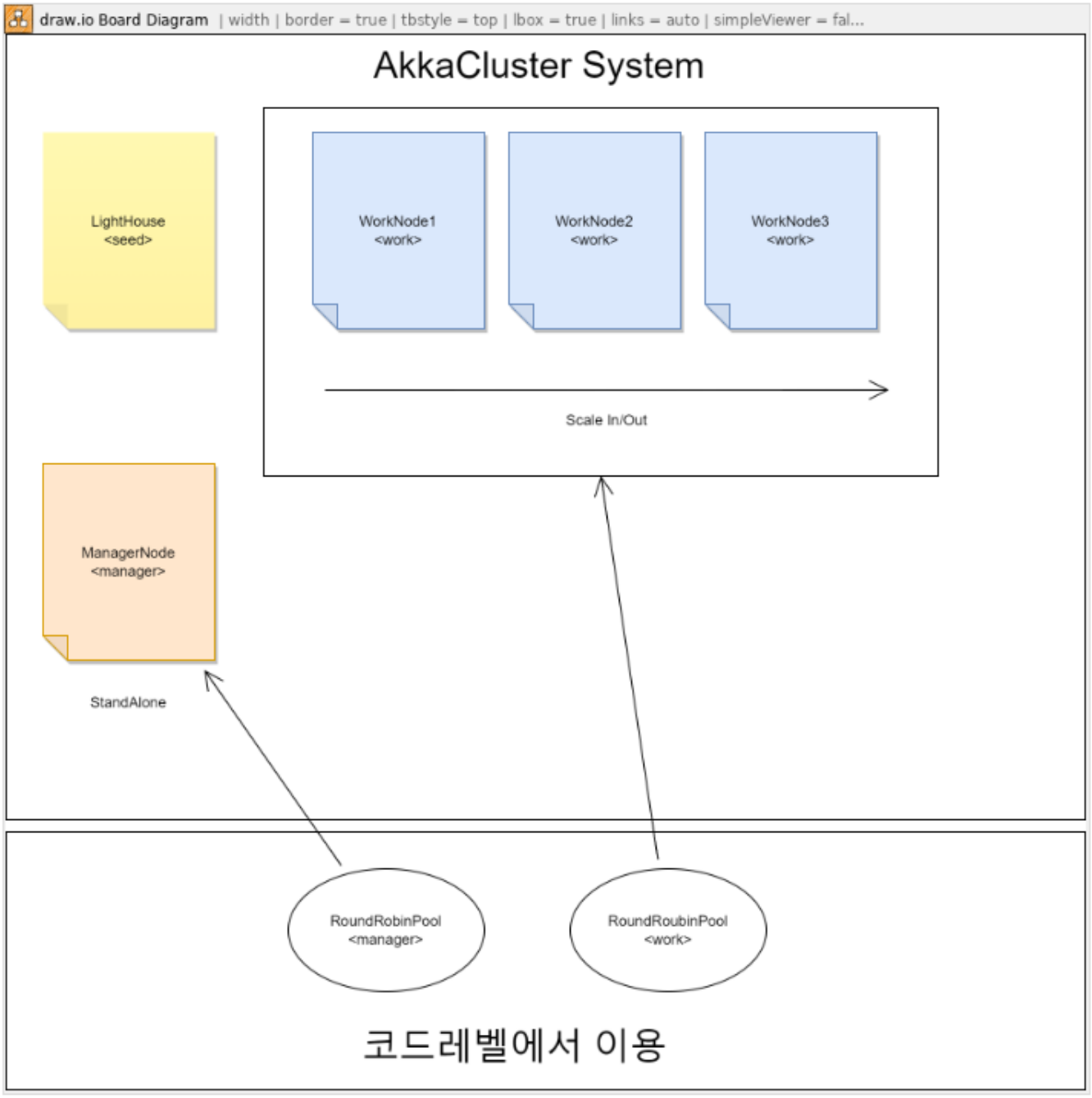


- LightHouse : ~ SeedNode .
  - Seed ( ) .
- Join : SeedNode Join .
- Gossip : Seed Discovery .
- Connect : Gossip .
  - P2P : Gossip Role .
- .net core CoreAPI .
  - NetCoreCluster
    - / ~ .

Role ~ Role

Role .

Cluster Role



- LightHouse(Seed) : Cluster Discovery .
- Role-Work : work , Role .
- Role-Manager : .
  - AkkaCluster SingleTone Cluster ~ .

(Location Transparency) .

WorkNode ~ .

```
        for(int i=0;i<testCount;i++){
            clusterActor.tell(testMessage + i , ActorRef.noSender());
        }
```

Nginnx LB .

(AKKA)

.

- :
- :
- :

`round-robin`	blocked URL ,
`broadcast`	blocked URL ,
`random`	
`consistent-hashing`	blocked URL hanshake LB-l7 AKKA Redis .

`tail-chopping`	, ( )
	:
	`within = 10s`
	`tail-chopping-router.interval = 20ms`

`scatter-gather`	blocked URL ,
`smallest-mailbox`	blocked URL ,
`scatter-gather`	blocked URL , : `within = 10s`

AKKA Router/Stream .

Akka Stream .

1. **(Back Pressure)** : .
2. : (Source), (Flow), (Sink) .
3. : Akka , .
4. : .
5. : .
6. : .
7. **Throttle** : , . , .

Akka Stream . Throttle , .

Spring Boot Cluster DockerCompose .

DockerCompose

```
version: '3.5'
services:
  light-house:
    image: registry.webnori.com/javalabs-lighthouse:dev
    ports:
      - "8081:8080"
    environment:
      TZ: Asia/Seoul
      akka.role: seed
      akka.seed: akka://ClusterSystem@light-house:12000
      akka.hostname: light-house
      akka.hostport: 12000
      akka.cluster-config: cluster.conf
    networks:
      - mynet
  work-node1:
    image: registry.webnori.com/javalabs-api:dev
    ports:
      - "8082:8080"
    depends_on:
      - light-house
    environment:
      TZ: Asia/Seoul
      akka.role: work
      akka.seed: akka://ClusterSystem@light-house:12000
      akka.hostname: work-node1
      akka.hostport: 12000
      akka.cluster-config: cluster.conf
    networks:
      - mynet
  work-node2:
    image: registry.webnori.com/javalabs-api:dev
    ports:
      - "8083:8080"
    depends_on:
      - light-house
    environment:
      TZ: Asia/Seoul
      akka.role: work
      akka.seed: akka://ClusterSystem@light-house:12000
      akka.hostname: work-node2
      akka.hostport: 12000
      akka.cluster-config: cluster.conf
    networks:
      - mynet
  manager-node:
    image: registry.webnori.com/javalabs-api:dev
    ports:
      - "8084:8080"
    depends_on:
      - light-house
    environment:
      TZ: Asia/Seoul
      akka.role: manager
      akka.seed: akka://ClusterSystem@light-house:12000
      akka.hostname: manager-node
      akka.hostport: 12000
      akka.cluster-config: cluster.conf
    networks:
      - mynet
networks:
  mynet:
    driver: bridge
```

- 8080 : spring boot web api .
- 12000 : akka cluster listen .
- akka cluster
    - role : role. role . .
    - seed : seed light-house .

- hostname : hostname . ip .
- hostport :
- cluster-config : // .

cluster cluster.conf base .

Cluster - cluster.conf

```
akka{
  actor {
    provider = cluster
  }

  remote.artery {
    canonical {
      hostname = "127.0.0.1"
      port = 12551
    }
  }

  cluster {
    seed-nodes = [
      "akka://ClusterSystem@127.0.0.1:12551"
    ]
    role{
      seed.min-nr-of-members=1
    }

    # auto downing is NOT safe for production deployments.
    # you may want to use it during development, read more about it in the docs.
    #
    auto-down-unreachable-after = 10s
    downing-provider-class = "akka.cluster.sbr.SplitBrainResolverProvider"
  }

  extensions=["akka.cluster.metrics.ClusterMetricsExtension"]
}
```

.

() AKKA .

Cluster Split Brain

SplitBrain / , .

Akka Cluster "split brain" .

Split brain (network partitioning) (sub-clusters) . , .

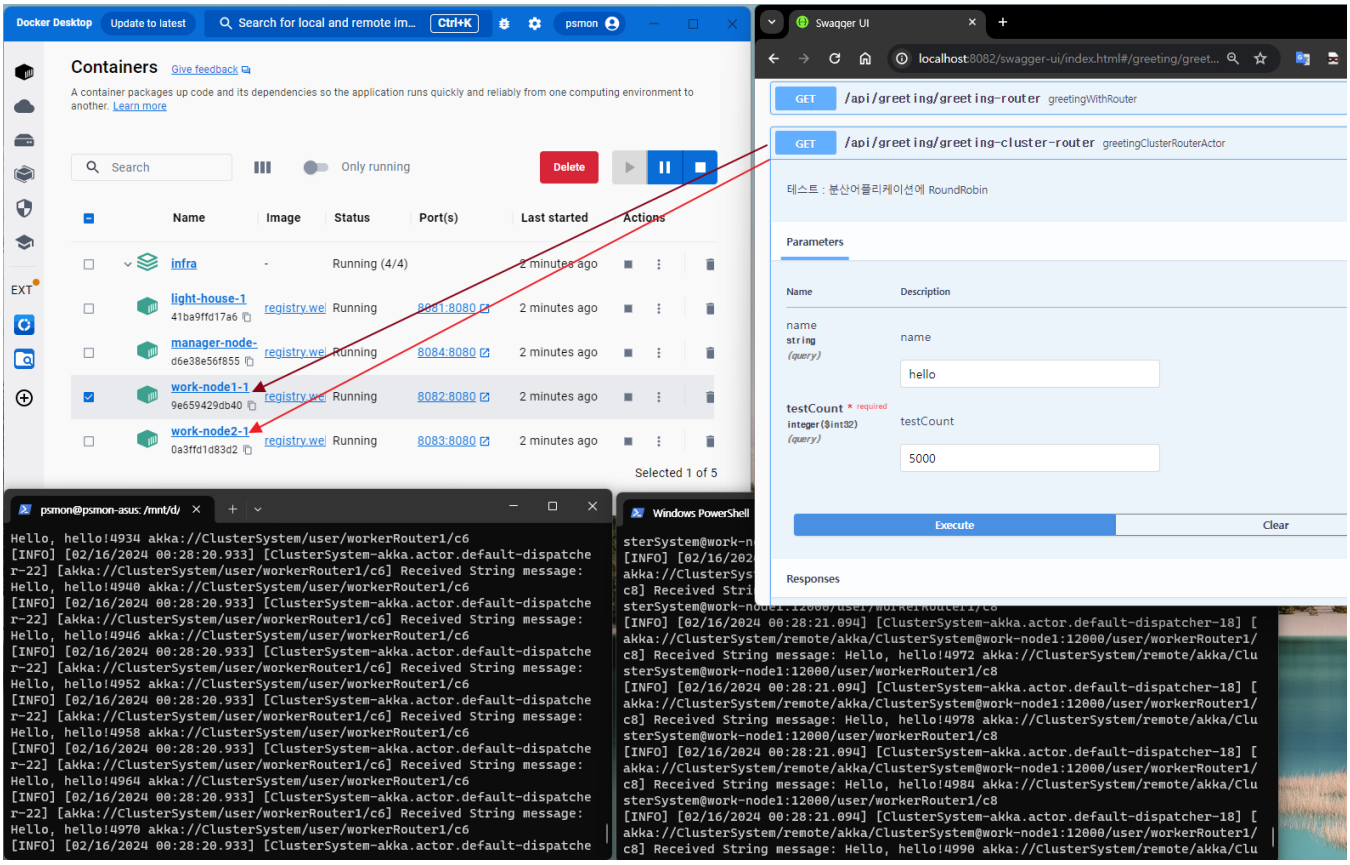
.

1. **Static Quorum**
   - : . .
   - : , .
2. **Keep Majority**
   - : , . " .
   - : , .
3. **Keep Oldest**
   - : . .
   - : (:) .
4. **Down All**
   - : . .
   - : , .
5. **Lease Majority**

- : (:) 'lease'() " . .
- : , .

Docker + Swagger TEST .

TEST by Docker



-
  - https://github.com/psmon/java-labs/blob/master/infra/docker-compose-cluster-local.yml
    - app .
    - javalabs\infra>docker-compose -f docker-compose-cluster-local.yml up -d

Spring Boot AkkaSystem

ApplicationStartup.java

```
@Component
public class ApplicationStartup implements ApplicationListener<ApplicationReadyEvent> {

    /**
     * This event is executed as late as conceivably possible to indicate that
     * the application is ready to service requests.
     */
    @SneakyThrows
    @Override
    public void onApplicationEvent(final ApplicationReadyEvent event) {
        AkkaManager.getInstance();
    }
}
```

- AkkaManager SpringBoot AkksSystem .

Actor

AkkaManager.java

```
//   :
// Actor ,  Spring
// Spring Bean   : https://www.baeldung.com/akka-with-spring
public final class AkkaManager {
    private static AkkaManager INSTANCE;
    @Getter
    private final ActorSystem actorSystem;
    private String akkaConfig;
    private String role;

    private String hostname;

    private String hostport;

    private String seed;


    @Getter
    private ActorRef greetActor;

    @Getter
    private ActorRef routerActor;

    @Getter
    private ActorRef clusterActor;

    @Getter
    private ActorRef clusterManagerActor;

    private AkkaManager() {

        akkaConfig = System.getenv("akka.cluster-config");
        role = System.getenv("akka.role");
        hostname = System.getenv("akka.hostname");
        hostport = System.getenv("akka.hostport");
        seed = System.getenv("akka.seed");

        actorSystem = serverStart("ClusterSystem", akkaConfig, role);

        InitActor();
    }

    public static AkkaManager getInstance() {
```

```
    if (INSTANCE == null) {
        INSTANCE = new AkkaManager();
    }
    return INSTANCE;
}

boolean isEmptyString(String string) {
    return string == null || string.isEmpty();
}

private ActorSystem serverStart(String sysName, String clusterConfig, String role) {

    Config regularConfig = ConfigFactory.load();

    Config combined;

    Boolean isCluster = !isEmptyString(clusterConfig) || !isEmptyString(role) || !isEmptyString(hostname)
            || !isEmptyString(hostport) || !isEmptyString(seed);

    if (isCluster) {
        Config newConfig = ConfigFactory.parseString(
                String.format("akka.cluster.roles = [%s]", role)).withFallback(
                ConfigFactory.load(clusterConfig));

        newConfig = ConfigFactory.parseString(
                String.format("akka.cluster.seed-nodes  = [\"%s\"] ", seed)).withFallback(
                ConfigFactory.load(newConfig));

        newConfig = ConfigFactory.parseString(
                String.format("akka.remote.artery.canonical.hostname = \"%s\" ", hostname)).withFallback(
                ConfigFactory.load(newConfig));

        newConfig = ConfigFactory.parseString(
                String.format("akka.remote.artery.canonical.port = %s ", hostport)).withFallback(
                ConfigFactory.load(newConfig));

        combined = newConfig
                .withFallback(regularConfig);
    } else {
        final Config newConfig = ConfigFactory.parseString(
                String.format("akka.cluster.roles = [%s]", "seed")).withFallback(
                ConfigFactory.load("cluster"));
        combined = newConfig
                .withFallback(regularConfig);
    }

    ActorSystem serverSystem = ActorSystem.create(sysName, combined);
    serverSystem.actorOf(Props.create(ClusterListener.class), "clusterListener");
    return serverSystem;
}

private void InitActor() {
    // Create Some Actor
    greetActor = actorSystem.actorOf(HelloWorld.Props()
            .withDispatcher("my-dispatcher"), "HelloWorld");

    // Create Router Actor
    routerActor = actorSystem.actorOf(new RoundRobinPool(5)
            .props(HelloWorld.Props()), "roundRobinPool");

    actorSystem.actorOf(TimerActor.Props()
            .withDispatcher("my-blocking-dispatcher"), "TimerActor");


    // Cluster Actor
    int totalInstances = 100;
    int maxInstancesPerNode = 3;
    boolean allowLocalRoutees = true;

    Set<String> useRoles = new HashSet<>(Arrays.asList("work"));
    clusterActor =
```

```
                actorSystem
                        .actorOf(
                                new ClusterRouterPool(
                                        new RoundRobinPool(0),
                                        new ClusterRouterPoolSettings(
                                                totalInstances, maxInstancesPerNode, allowLocalRoutees,
useRoles))
                                        .props(Props.create(ClusterHelloWorld.class)),
                                "workerRouter1");

        Set<String> useManagerRoles = new HashSet<>(Arrays.asList("manager"));
        clusterManagerActor =
                actorSystem
                        .actorOf(
                                new ClusterRouterPool(
                                        new RoundRobinPool(0),
                                        new ClusterRouterPoolSettings(
                                                totalInstances, maxInstancesPerNode, allowLocalRoutees,
useManagerRoles))
                                        .props(Props.create(ClusterHelloWorld.class)),
                                "workerRouter2");
    }

}
```

- Spring Boot DI SpringBoot .
- serverStart() : AkkaSystem .
- InitActor : .

ClusterHelloWorld.java

```
public class ClusterHelloWorld extends AbstractActor {
    private final LoggingAdapter log = Logging.getLogger(getContext().getSystem(), this);

    Cluster cluster = Cluster.get(getContext().system());

    public static Props Props() {
        return Props.create(ClusterHelloWorld.class);
    }

    //subscribe to cluster changes
    @Override
    public void preStart() {
        cluster.subscribe(self(), (ClusterEvent.SubscriptionInitialStateMode) ClusterEvent.
initialStateAsEvents(),
                ClusterEvent.MemberEvent.class, ClusterEvent.UnreachableMember.class);
    }

    @Override
    public void postStop() {
        cluster.unsubscribe(self());
    }

    @Override
    public Receive createReceive() {
        return receiveBuilder().match(ClusterEvent.MemberUp.class, mUp -> {
            log.info("Member is Up: {}", mUp.member());
        }).match(ClusterEvent.UnreachableMember.class, mUnreachable -> {
            log.info("Member detected as unreachable: {}", mUnreachable.member());
        }).match(ClusterEvent.MemberRemoved.class, mRemoved -> {
            log.info("Member is Removed: {}", mRemoved.member());
        }).match(ClusterEvent.MemberEvent.class, message -> {
        //
        }).match(String.class, s -> {
            log.info("Received String message: {} {}", s, context().self().path());
        })
        .match(TestClusterMessages.Ping.class, s -> {
            log.info("Received Ping message: {}",  context().self().path());
        })
        .build();
    }
}
```

- : Cluster / . .
- : Meber . .

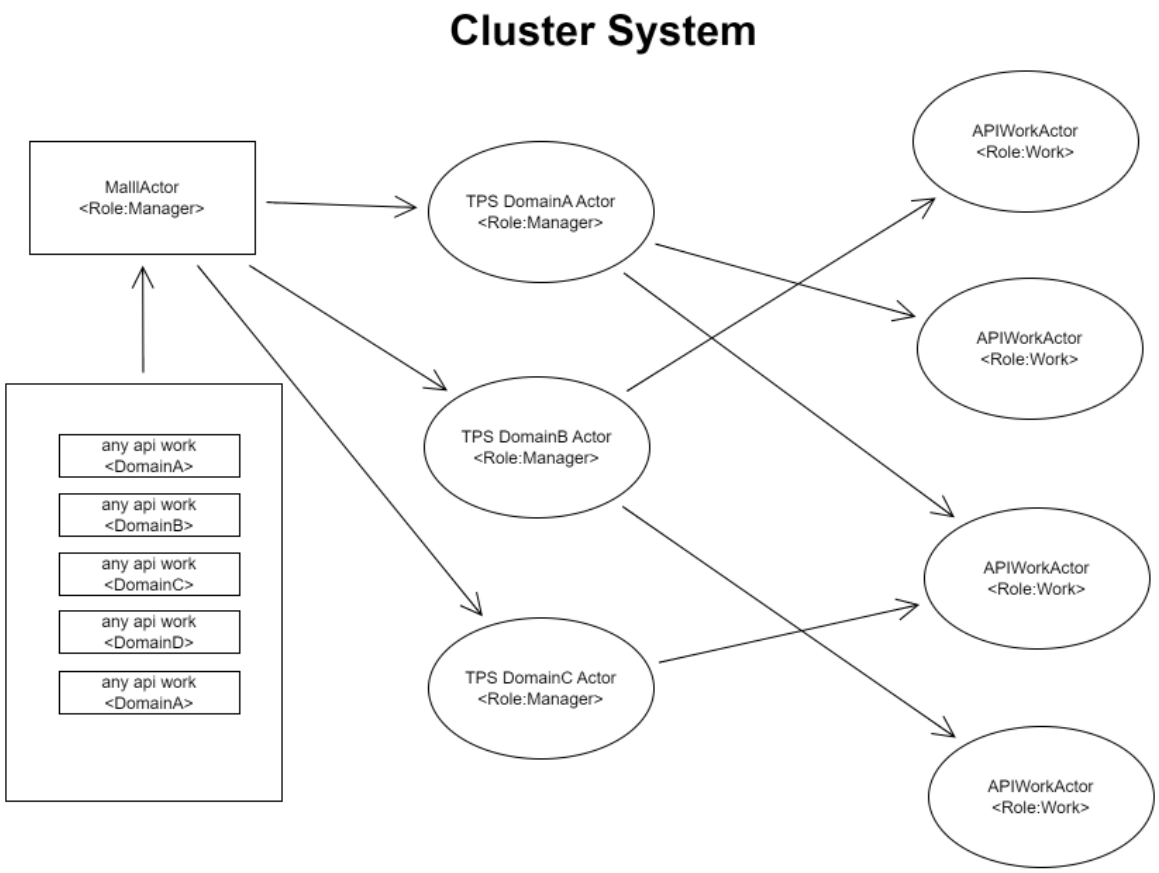
TPS

.

- Role
  - ,
- TPS
- Round Robind

.

- TPS .



TPS .

.

- API

```
@SpringBootTest
public class FactorialTest {

    private static Logger logger = LoggerFactory.getLogger(FactorialTest.class);
    private static ActorSystem clusterSystem1;
    private static ActorSystem clusterSystem2;
    private static ActorSystem clusterSystem3;

    private int maxServerUptime = 20;

    private static ActorSystem serverStart(String sysName, String config, String role) {
        final Config newConfig = ConfigFactory.parseString(
                String.format("akka.cluster.roles = [%s]", role)).withFallback(
                ConfigFactory.load(config));

        ActorSystem serverSystem = ActorSystem.create(sysName, newConfig);
        serverSystem.actorOf(Props.create(ClusterListener.class), "clusterListener");
        return serverSystem;
    }

    @BeforeClass
    public static void setup() {
        // Seed
        clusterSystem1 = serverStart("ClusterSystem", "server", "seed");
```

```
        // Works Nodes
        clusterSystem2 = serverStart("ClusterSystem", "factorial", "backend");
        clusterSystem2.actorOf(Props.create(FactorialBackend.class), "factorialBackend");

        clusterSystem3 = serverStart("ClusterSystem", "factorial", "backend");
        clusterSystem3.actorOf(Props.create(FactorialBackend.class), "factorialBackend");

        logger.info("========= sever loaded =========");
    }

    @AfterClass
    public static void gracefulDown() {
        clusterSystem3.terminate();
        clusterSystem2.terminate();
        clusterSystem1.terminate();
        logger.info("========= sever down =========");
    }

    @Test
    public void clusterTest() {
        logger.info("========= client start =========");
        final int upToN = 200;

        final Config config = ConfigFactory.parseString(
                "akka.cluster.roles = [client]").withFallback(
                ConfigFactory.load("factorial"));

        final ActorSystem system = ActorSystem.create("ClusterSystem", config);
        system.log().info("Factorials will start when 2 backend members in the cluster.");

        new TestKit(system) {
            {
                ActorRef probe = getRef();
                Cluster.get(system).registerOnMemberUp(new Runnable() {
                    @Override
                    public void run() {
                        ActorRef frontActor = system.actorOf(Props.create(FactorialClient.class, upToN, false),
                                "factorialClient");
                        frontActor.tell(new FactorialRequest(upToN), probe);
                    }
                });
                expectMsgClass(Duration.ofSeconds(maxServerUptime), FactorialResult.class);
            }
        };
    }
}
```

N .

.

Spring Boot

- Seed LightHouse
  - SRC : https://github.com/psmon/java-labs/tree/master/lighthouse
- Spring BOOT ActorSystem
  - https://github.com/psmon/java-labs/blob/master/springweb/src/main/java/com/webnori/springweb/example/akka/AkkaManager.java
  - https://github.com/psmon/java-labs/blob/master/springweb/src/main/java/com/webnori/springweb/example/akka/actors/cluster/ClusterHelloWorld.java
  - https://github.com/psmon/java-labs/blob/master/springweb/src/main/resources/application.conf
  - https://github.com/psmon/java-labs/blob/master/springweb/src/main/resources/cluster.conf
  - https://github.com/psmon/java-labs/blob/master/infra/docker-compose-cluster-local.yml
  - https://github.com/psmon/java-labs/tree/master/springweb/src/test/java/com/webnori/springweb/akka

Akka .

.

- - API
- [Labs] AKKA
- API

Next

docker-compose .

Spring AkkaCluster RKE- .

RestAPI StateLess POD LB Ingress

Discovery

.

- https://wiki.webnori.com/display/rancher