DISTRIBUTED SYSTEMS

Lab 1

João Leitão, Sérgio Duarte, Pedro Camponês

GOAL

In the end of this lab you should be able to:

- Use maven to compile, assembly and create a docker image
- Understand how docker works
- Use multicast to discover (the URL of web) servers in Java

GOAL

In the end of this lab you should be able to:

- **Use maven to compile, assembly and create a docker image**
- Understand how docker works
- Use multicast to discover (the URL of web) servers in Java

BUILDING TOOLS

Maven is a software project management tool used for building Java projects.

Simplifies the use of dependencies needed by a program.

We will be using maven for building all projects in this course.

When using your preferred IDE, make sure you import the code provided as a Maven project.

POM.XML – CONFIGURATION FILE

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>sd2425</groupId>
        <artifactId>sd2425-lab1</artifactId>
        <version>1.0</version>
        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <authors>xxxxx-yyyyy</authors>
        </properties>
```

This property will be used to name the **docker** container image. Change it for the numbers of your group.

POM.XML – CONFIGURATION FILE (CONT)

```
<build>
        <sourceDirectory>src</sourceDirectory>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
```

Allows to define the source directory within the project

Allows to define the java version to use

POM.XML – CONFIGURATION FILE (CONT)

```
<plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
                <archive>
                </archive>
                <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
        </configuration>
</plugin>
```

Used to create a single file with all code.

POM.XML – CONFIGURATION FILE (CONT)

This plugin allows to create a docker image containing the compiled jar in this project (using the local Dockerfile).

The name of the created container depends on the property defined before.

```
<plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>docker-maven-plugin</artifactId>
        <version>0.44.0</version>
        <configuration>
                <images>
                    <image>
                        <name>${project.artifactId}-${authors}</name>
                        <build>
                                <dockerFile>${project.basedir}/Dockerfile</dockerFile>
                        </build>
                     </image>
                </images>
        </configuration>
</plugin>
```

RUNNING MAVEN

mvn clean - cleans the project, removing generated files

mvn compile – compiles the project

mvn assembly:single – creates a single file with all compiled classes and dependencies

mvn docker:build – builds a docker image using the Dockerfile in the current directory.

Note: you can run all at once, by doing:

mvn clean compile assembly:single docker:build

GOAL

In the end of this lab you should be able to:

- Use maven to compile, assembly and create a docker image
- **Understand how docker works**
- Use multicast to discover servers in Java

DOCKER

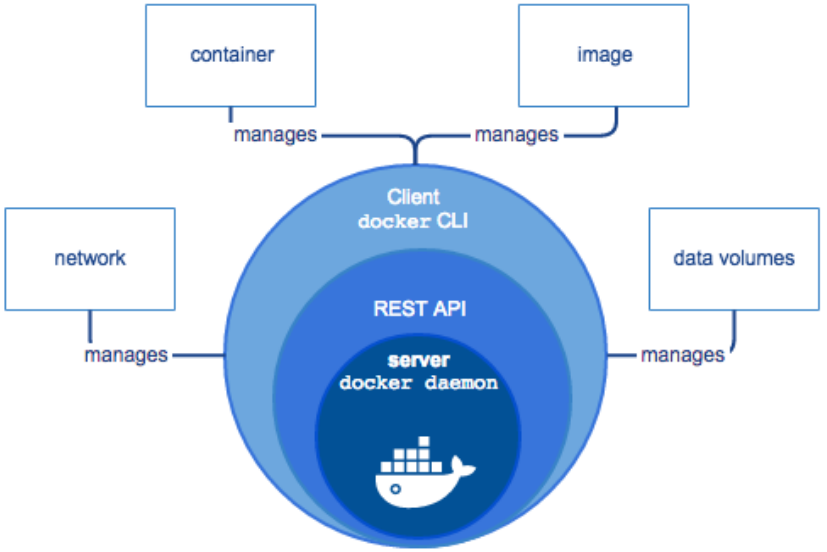
Docker is a system/platform for running applications using container technology.

A container includes all software necessary to run the application and each container executes isolated from the other containers.

Contrary to Virtual Machines a container does not run its own instance of the operating system (it used the functionalities of the OS provided by its execution environment).

DOCKER ENGINE

- Docker daemon (dockerd) manages Docker objects such as images, containers, networks, and volumes.
- The docker client (usually referred as docker cli) sends requests to docker daemon.

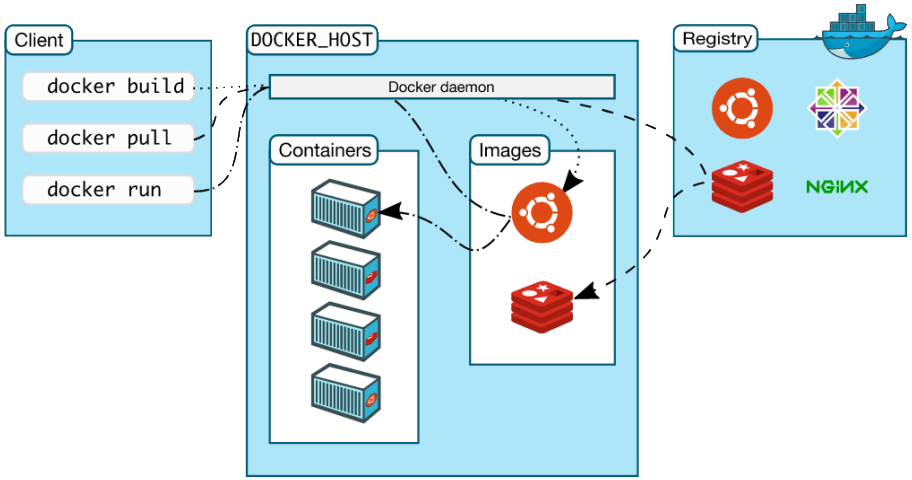


DOCKER ENGINE (2)

A Docker *registry* stores Docker images. Docker is configured to search in Docker Hub by default.

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization.

A Docker image can be created from the specification in a Dockerfile.



DOCKERFILE

```
# base image - an image with openjdk  17
FROM openjdk:17
# working directory inside docker image
WORKDIR /home/sd
# copy the jar created by assembly to the docker image
COPY target/sd2425-lab1-1.0.jar sd.jar
# copy a script that can be used to launch TCP client or server
COPY exec.sh exec.sh
# run Discovery when starting the docker image (for tests)
CMD ["java", "-cp", "/home/sd/sd.jar", "sd.lab1.Discovery"]
```

CREATING A CONTAINER IMAGE

With the provided maven project, to build the image based on the Dockerfile, run:

mvn docker:build

It is also possible to build the container image using the docker build command:

docker build -t name dir_of_dockerfile

docker build -t sd2425-lab1-xxxxx-yyyyy .

-t is used to define the name of the image.

DOCKER: USEFUL COMMANDS

Docker run command:

docker run [params] imagename [cmd]

Start an image and run the default command:

docker run sd2425-lab1-xxxxx-yyyyy

Start an image, but run an alternative command – e.g. the bash:

docker run -it sd2425-lab1-xxxxx-yyyyy /bin/bash

DOCKER NETWORKING

By default, all containers started in a machine will be able to connect to each other through a virtual network.

Each container is assigned an IP and a hostname. The hostname is only known locally. The hostname can be changed using the –h option as show below:

docker run -h myhostname sd2425-lab1-xxxxx-yyyyy

DOCKER NETWORKING (2)

It is possible to create a bridge network that connect containers in a machine with hostname resolution. To create a bridged network named *sdnet*, run:

docker network create -d bridge sdnet

When running the container, specify the network (--network *sdnet*), the name and hostname (--name srv1 --hostname srv1):

docker run -h srv1 --name srv1 --network sdnet sd2425-lab1-xxxxx-yyyyy

DOCKER: MORE USEFUL COMMANDS

docker ps [OPTIONS]

docker ps : Lists containers running.

docker ps –a : shows all containers including those that are stopped

docker exec [OPTIONS] CONTAINER cmd

Executes a command in a running image (e.g.:
docker exec -it 001b898b6d23 /bin/bash).

docker logs [OPTIONS] CONTAINER

Fetch the logs of a running container; -f options keeps connected (e.g.:
docker logs -f 001b898b6d23).

(this command is useful if the container was executed in background with the option –d on the command run)

DOCKER: MORE USEFUL COMMANDS

docker kill [OPTIONS] CONTAINER [CONTAINER...]

Kills one or more containers.

docker rm [OPTIONS] CONTAINER [CONTAINER...]

Cleans up one or more exited containers.

docker system prune

Cleans up all unused data (incl. exited containers).

DOCKER: MORE USEFUL COMMANDS (2)

docker images [OPTIONS] [REPOSITORY[:TAG]]

Lists images.

GOAL

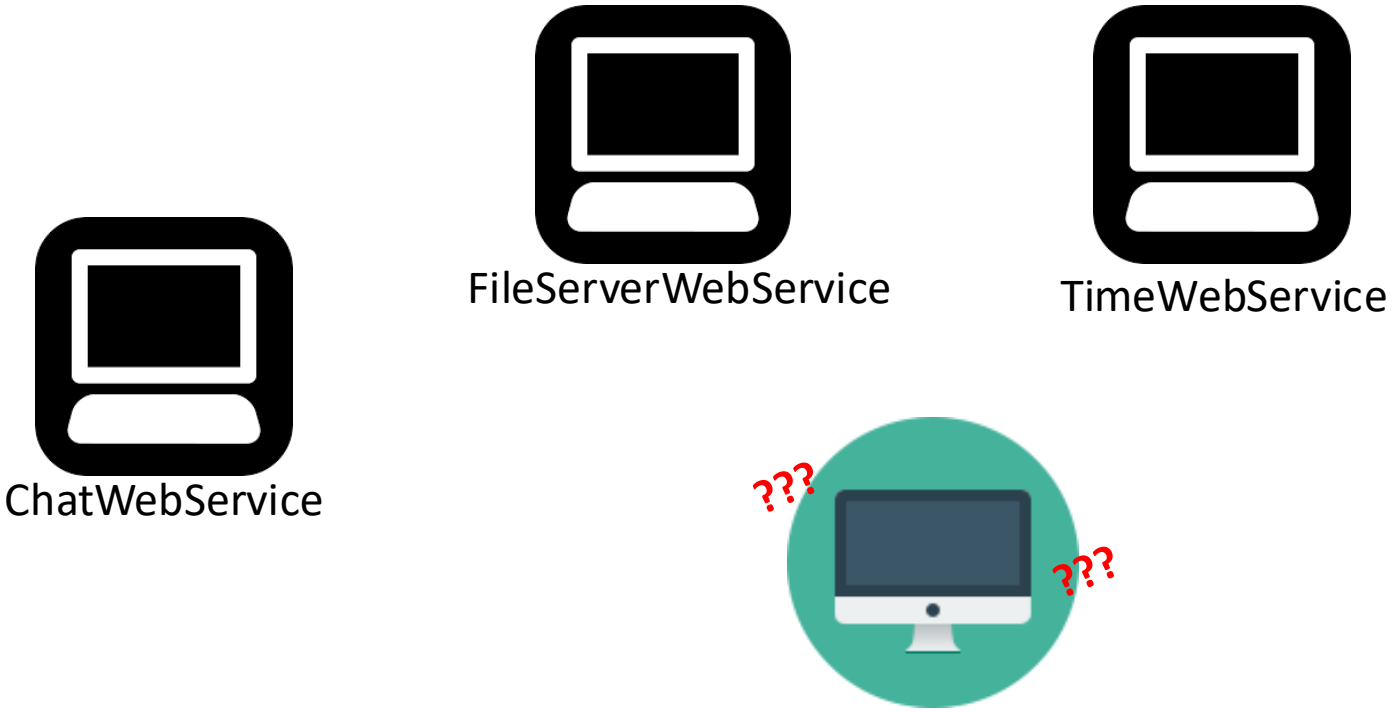
In the end of this lab you should be able to:

- Use maven to compile, assembly and create a docker image
- Understand how docker works
- **Use multicast to discover servers in Java**

How to perform service discovery?

How does a client discover a server?

How does a server discover other servers?



How to perform service discovery?

One solution is to use **IP Multicast**

(There are two flavors)

ChatWebService

FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

1st Alternative: Server Initiated

ChatWebService

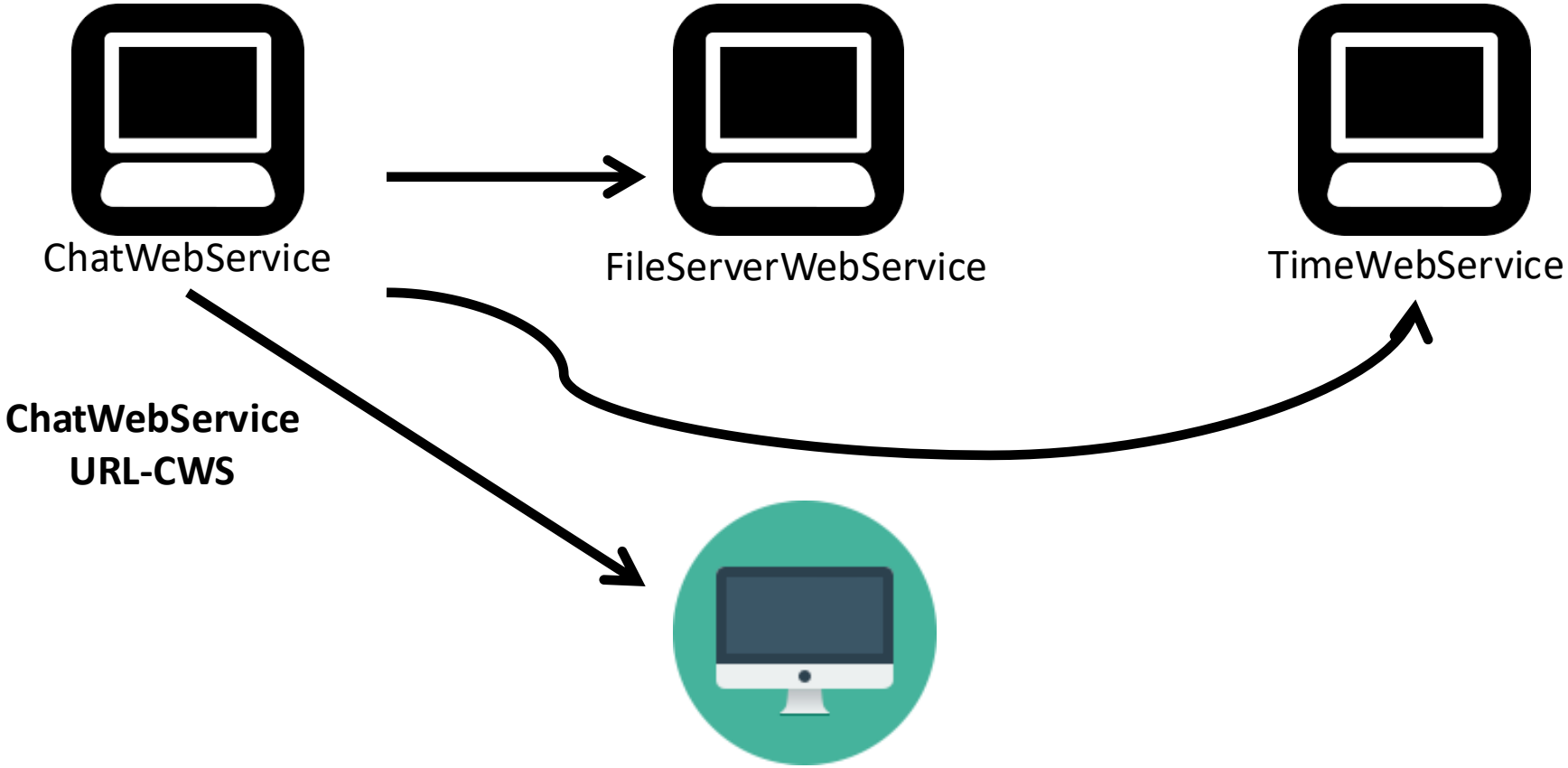
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

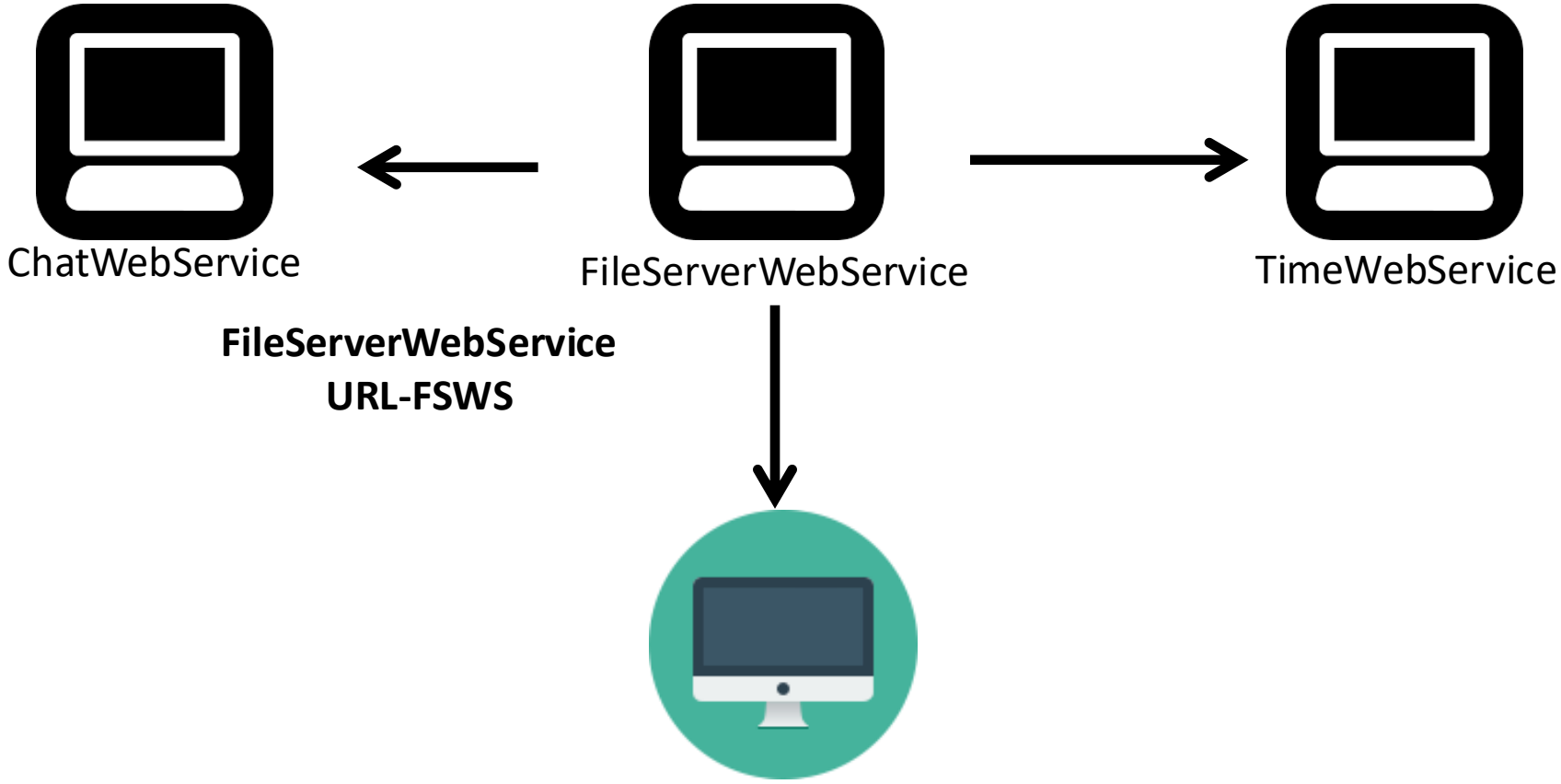
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

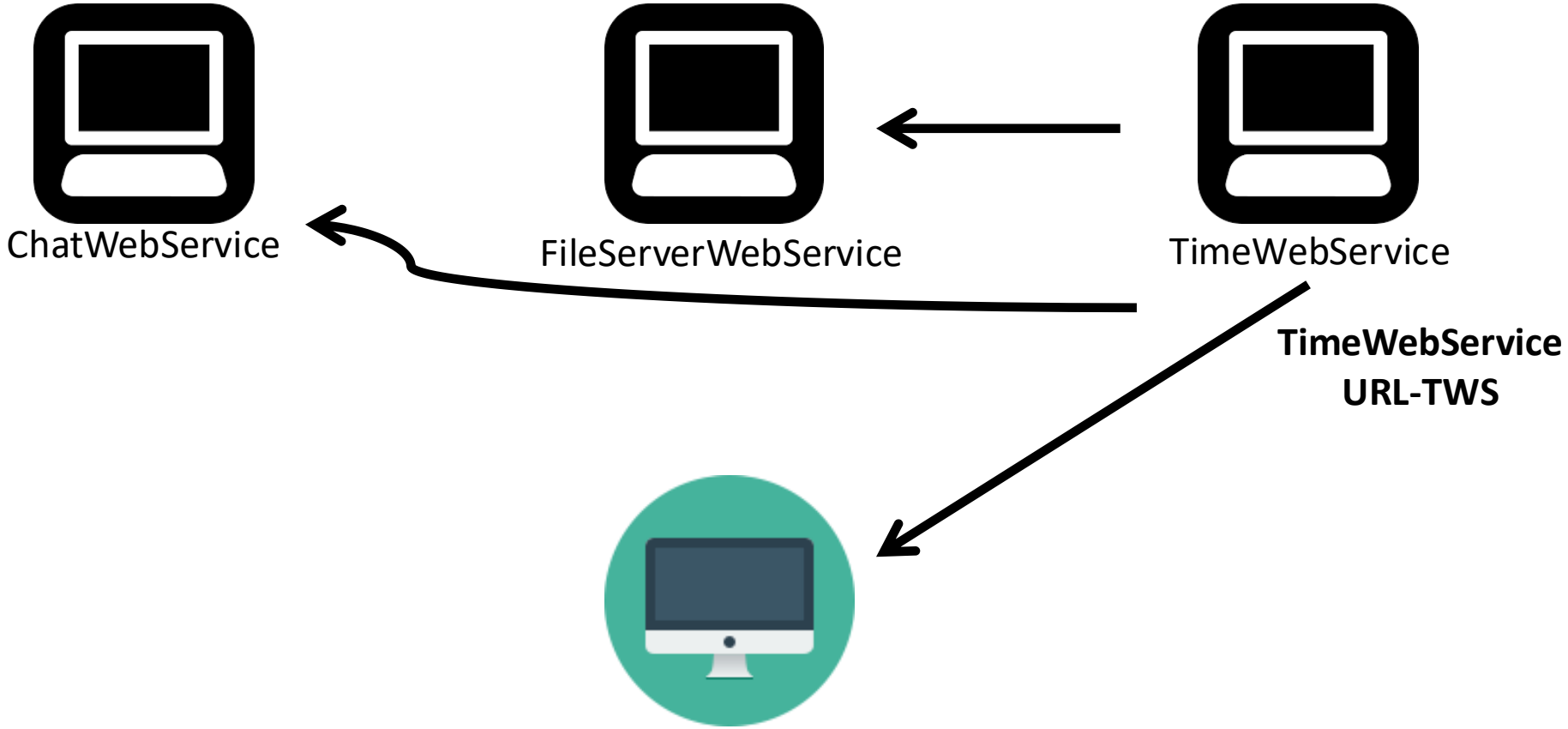
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

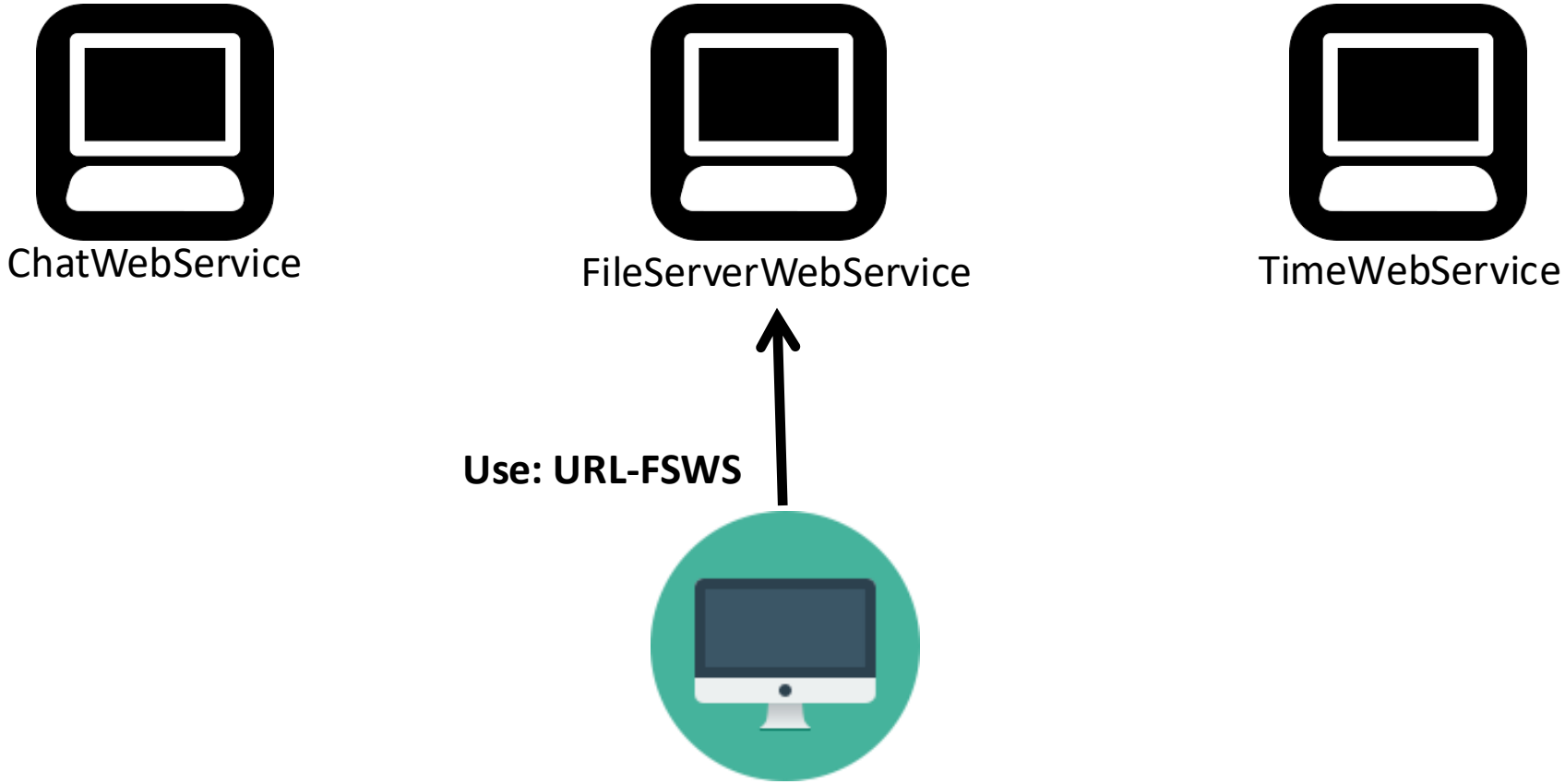
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

2nd Alternative: Client Initiated

ChatWebService

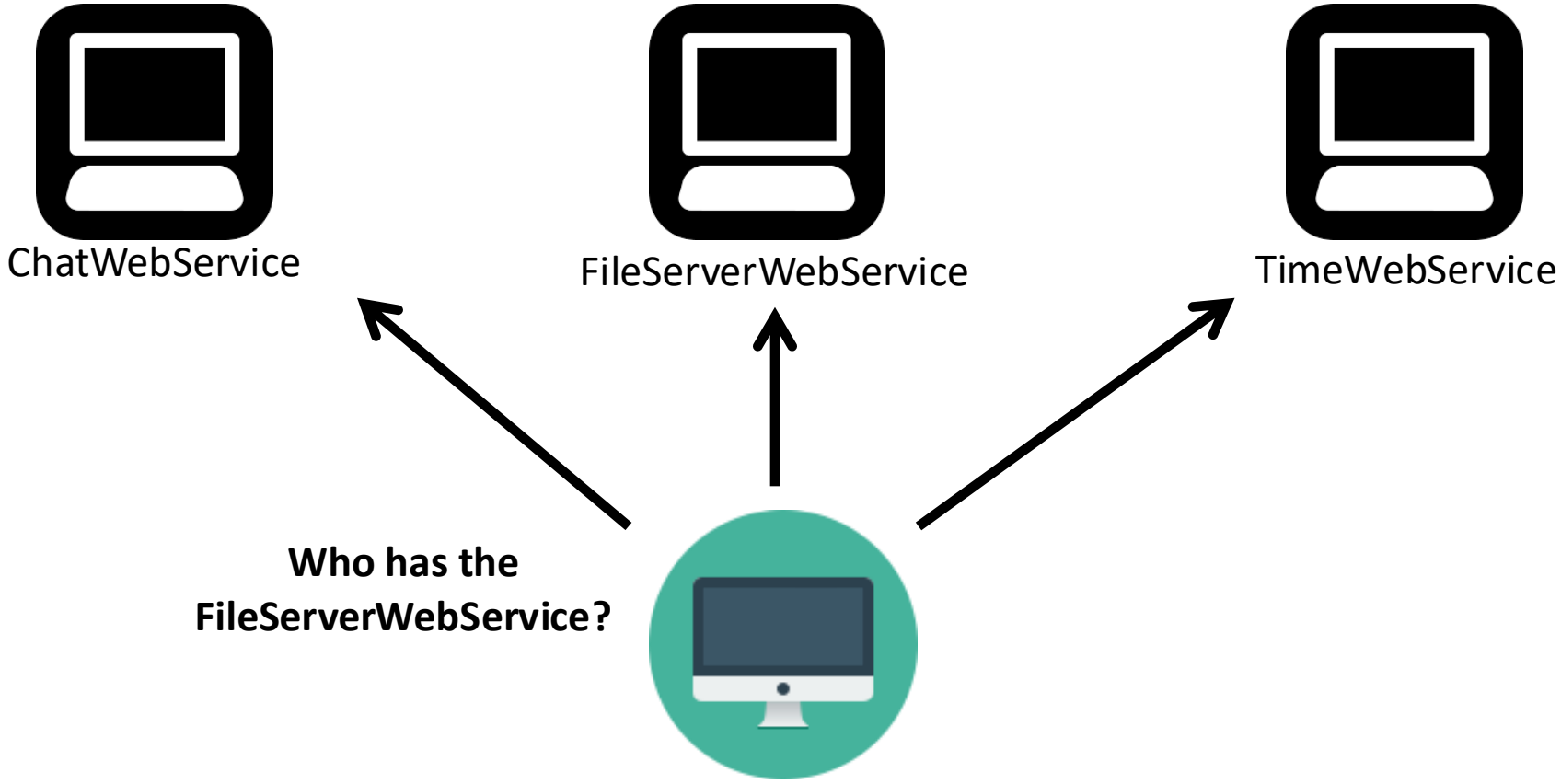
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

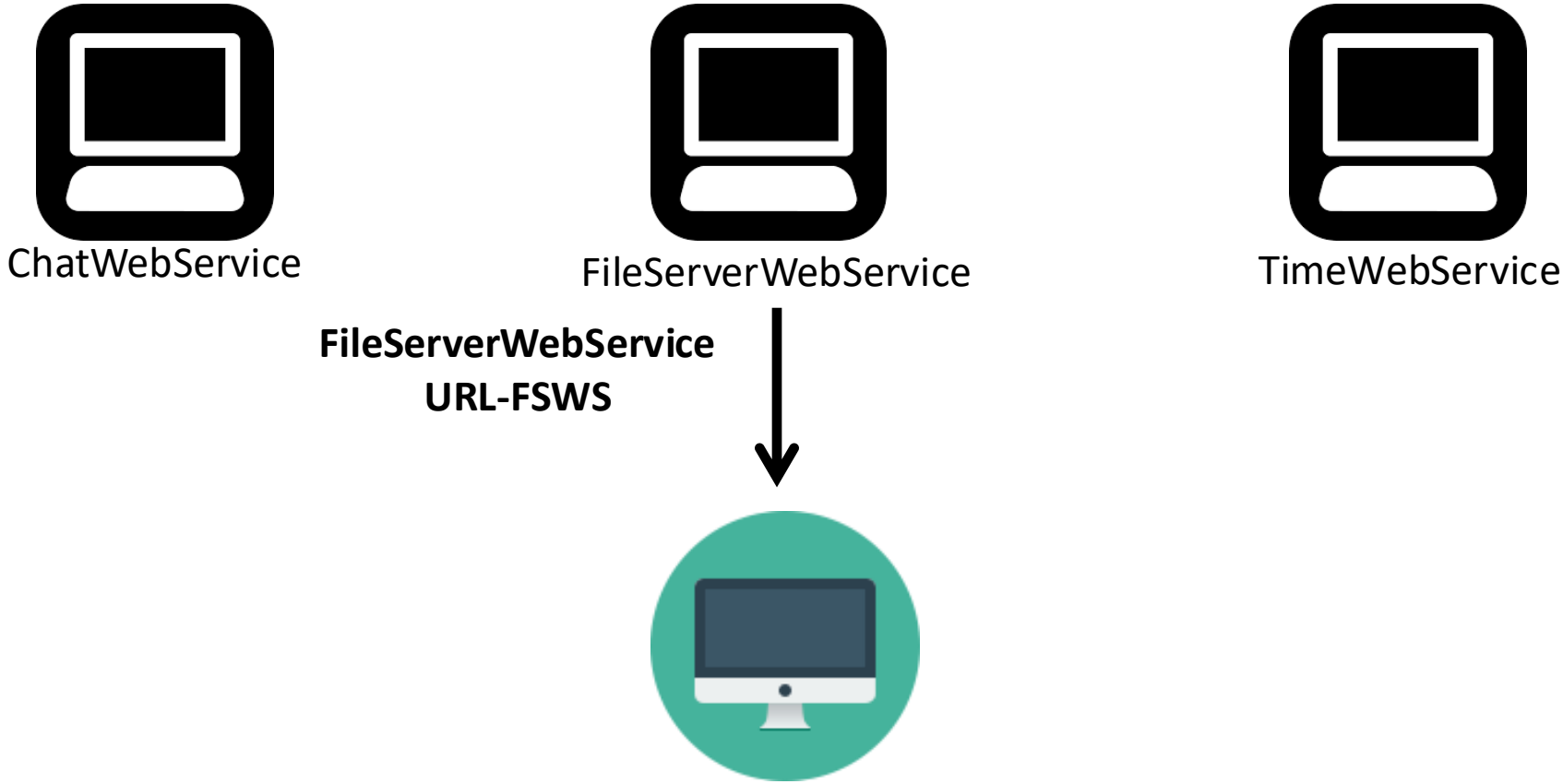
2nd Alternative: Client Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

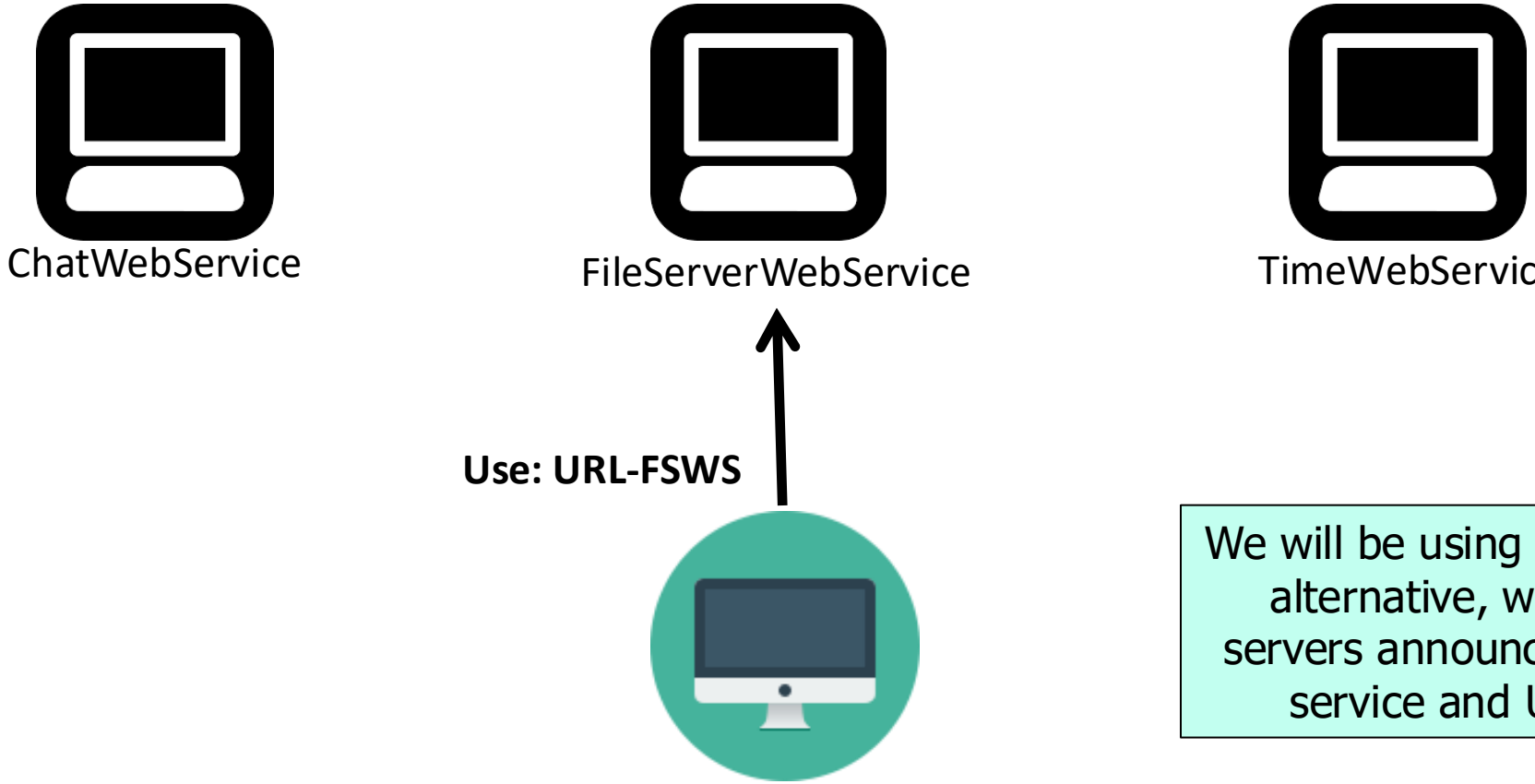
2nd Alternative: Client Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

2nd Alternative: Client Initiated



We will be using the first alternative, where servers announce their service and URL

SERVICE DISCOVERY (1: STATE AND INITIALIZATION)

```
private final InetSocketAddress addr;
private final String serviceName;
private final String serviceURI;
private final MulticastSocket ms;


Discovery(InetSocketAddress addr, String serviceName, String serviceURI) throws
SocketException, UnknownHostException, IOException {
   this.addr = addr;
   this.serviceName = serviceName;
   this.serviceURI = serviceURI;


   if (this.addr == null) {
      throw new RuntimeException("A multinet address has to be provided.");
   }
   this.ms = new MulticastSocket(addr.getPort());
   this.ms.joinGroup(addr, NetworkInterface.getByInetAddress(InetAddress.getLocalHost()));
}
```

SERVICE DISCOVERY (1: STATE AND INITIALIZATION)

```
private final InetSocketAddress addr;
private final String serviceName;
private final String serviceURI;
private final MulticastSocket ms;


Discovery(InetSocketAddress addr, String serviceName, String serviceURI) throws
SocketException, UnknownHostException, IOException {
   this.addr = addr;
   this.serviceName = serviceName;
   this.serviceURI = serviceURI;


   if (this.addr == null) {
      throw new RuntimeException("A multinet address
   }
   this.ms = new MulticastSocket(addr.getPort());
   this.ms.joinGroup(addr, NetworkInterface.getByInetAddress(InetAddress.getLocalHost()));
}
```

Create the multicast socket and join the group to receive messages.

SERVICE DISCOVERY (2: SENDING ANNOUNCEMENTS)

```
byte[] announceBytes = String.format("%s%s%s", serviceName, DELIMITER, serviceURI).getBytes();

DatagramPacket announcePkt = new DatagramPacket(announceBytes, announceBytes.length, addr);
try {
        // start thread to send periodic announcements
        new Thread(() -> {
                for (;;) {
                        try {
                                ms.send(announcePkt);
                                Thread.sleep(DISCOVERY_ANNOUNCE_PERIOD);
                        } catch (Exception e) {
                                e.printStackTrace(); // do nothing
                        }
                }
        }).start();
                } catch (Exception e) {
                        e.printStackTrace();
        }
```

SERVICE DISCOVERY (2: SENDING ANNOUNCEMENTS)

```
byte[] announceBytes = String.format("%s%s%s", serviceName, DELIMITER, serviceURI).getBytes();

DatagramPacket announcePkt = new DatagramPacket(announceBytes, announceBytes.length, addr);
try {
        // start thread to send periodic announcements
        new Thread(() -> {
                for (;;) {
                        try {
                                ms.send(announcePkt);
                                Thread.sleep(DISCOVERY_ANNOUNCE_PERIOD);
                        } catch (Exception e) {
                                e.printStackTrace(); // do nothing
                        }
                }
        }).start();
                } catch (Exception e) {
                        e.printStackTrace();
        }
```

Periodically send the announcement message.

SERVICE DISCOVERY (3: RECEIVING ANNOUNCEMENTS)

```
// start thread to collect announcements received from the network.
new Thread(() -> {
  DatagramPacket pkt = new DatagramPacket(new byte[MAX_DATAGRAM_SIZE], MAX_DATAGRAM_SIZE);
    for (;;) {
      try {
        pkt.setLength(MAX_DATAGRAM_SIZE);
        ms.receive(pkt);
        String msg = new String(pkt.getData(), 0, pkt.getLength());
        String[] msgElems = msg.split(DELIMITER);
        if (msgElems.length == 2) { // periodic announcement
          System.out.printf("FROM %s (%s) : %s\n", pkt.getAddress().getHostName(),
pkt.getAddress().getHostAddress(), msg);
          // TODO: to complete by recording the received information
        }
      } catch (IOException e) { // do nothing }
    }
}).start();
```

SERVICE DISCOVERY (3: RECEIVING ANNOUNCEMENTS)

```
// start thread to collect announcements received from the network.
new Thread(() -> {
  DatagramPacket pkt = new DatagramPacket(new byte[MAX_DATAGRAM_SIZE], MAX_DATAGRAM_SIZE);
    for (;;) {
      try {
        pkt.setLength(MAX_DATAGRAM_SIZE);
        ms.receive(pkt);
        String msg = new String(pkt.getData(), 0, pkt.getLength());
        String[] msgElems = msg.split(DELIMITER);
        if (msgElems.length == 2) { // periodic announcement
          System.out.printf("FROM %s (%s) : %s\n", pkt.getAddress().getHostName(),
pkt.getAddress().getHostAddress(), msg);
          // TODO: to complete by recording the received information
        }
      } catch (IOException e) { // do nothing }
    }
}).start();
```

Receive and process message.

How to integrate the Discovery mechanism?

Considering a server and a client model (simple TCP server).

- **Server:**
  - Instantiates the Discovery providing the multicast group address and registering its name and its Uniform Resource Identifier (URI)
  - Starts the Discovery process to start its own announcements.
- **Client:**
  - Instantiates the Discovery only providing the multicast address group and starts the Discovery to collect and register announcement in the network.
  - Queries the Discovery for the target service name to obtain a URI of the server and then connects to the server.

SIMPLE TCP SERVER

```
public static void main(String[] args) throws Exception {
//TODO: Use Discovery to announce the uri of this server, in the form of (tcp://hostname:port)
//Create a server socket and wait for incoming TCP connections from client
  try(ServerSocket ssocket = new ServerSocket( PORT )) {
    Log.info("My IP address is: " + InetAddress.getLocalHost().getHostAddress());
    Log.info("Accepting connections at: " + ssocket.getLocalSocketAddress() ) ;
    while( true ) {
       Socket csocket = ssocket.accept() ;
       System.err.println("Accepted connection from client at: " + csocket.getRemoteSocketAddress() ) ;
       int n;
       byte[] buf = new byte[ BUF_SIZE];

       //Until the connection is closed receive lines of text from the client and print them out
       while( (n = csocket.getInputStream().read(buf)) > 0)
         System.out.write( buf, 0, n);
       Log.info("Connection closed.") ;
    }
  }
}
```

SIMPLE TCP SERVER

```
public static void main(String[] args) throws Exception {
//TODO: Use Discovery to announce the uri of this server, in the form of (tcp://hostname:port)
//Create a server socket and wait for incoming TCP connections from client
  try(ServerSocket ssocket = new ServerSocket( PORT )) {
    Log.info("My IP address is: " + InetAddress.getLocalHost().getHostAddre
    Log.info("Accepting connections at: " + ssocket.getLocalSocketAddress() ) ;
    while( true ) {
       Socket csocket = ssocket.accept() ;
       System.err.println("Accepted connection from client at: " + csocket.getRemoteSocketAddress() ) ;
       int n;
       byte[] buf = new byte[ BUF_SIZE];

       //Until the connection is closed receive lines of text from the client and print them out
       while( (n = csocket.getInputStream().read(buf)) > 0)
         System.out.write( buf, 0, n);
       Log.info("Connection closed.") ;
    }
  }
}
```

When starting the server prints its own IP address to standard output.

SIMPLE TCP CLIENT

```
public static void main(String[] args) throws Exception {
   //TODO: Change to use the Discovery to obtain the hostname and port of the server (format
tpc://hostname:port;
   Scanner scan = new Scanner(System.in);
   String serverAddr = scan.nextLine();

   //Process the string with the server address to extract IP address and port
   String[] elements = serverAddr.split(":");
   int port = Integer.parseInt(elements[2]);
   String hostname = elements[1].replaceAll("//", "");
   //Establish a TCP connection to the server and send lines of text from standard input until input is !quit
   try( Socket sock = new Socket( hostname, port)) {
     String input;
     do {
        input = scan.nextLine();
        sock.getOutputStream().write( (input + System.lineSeparator()).getBytes() );
     } while( ! input.equals(QUIT));
   }
   scan.close();
}
```

SIMPLE TCP CLIENT

```
public static void main(String[] args) throws Exception {

   //TODO: Change to use the Discovery to obtain the hostname and port of the server (format
tpc://hostname:port;

   Scanner scan = new Scanner(System.in);
   String serverAddr = scan.nextLine();


   //Process the string with the server address to extract IP address and port
   String[] elements = serverAddr.split(":");
   int port = Integer.parseInt(elements[2]);
   String hostname = elements[1].replaceAll("//", "");
   //Establish a TCP connection to the server and send lines of text from standard input until input is !quit
   try( Socket sock = new Socket( hostname, port)) {
     String input;
     do {
        input = scan.nextLine();
        sock.getOutputStream().write( (input + System.lineSeparator()).getBytes() );
     } while( ! input.equals(QUIT));
   }
   scan.close();
}
```

This is collecting the URI from the user. You should be able to use the Discovery instead.

EXERCISE

1. Run multiple container images and verify that each container will receive announcement from its own and other containers (using the main in the Discovery class).
2. Complete the code to record information about running services. Suggestion: store the time of received announcement to know which servers are currently reachable.
3. Modify the TcpServer and TcpClient to use the Discovery.

NOTE: this code will be used as the basis to discover servers in your project.

EXERCISE

1. Run multiple container images and verify that each container will receive announcement from its own and other containers (using the main in the Discovery class).

To start a container (that by default runs the main on the Discovery class) you can use multiple times:

```
docker run sd2425-lab1-xxxxx-yyyyy
```

Optionally you can attribute a hostname and name with options -h and --name

EXERCISE

3. Modify the TcpServer and TcpClient to use the Discovery.

To start a container (that starts the TCP client or server respectively) you can take advantage of the provided exec.sh script:

```
docker run -it sd2425-lab1-xxxxx-yyyyy bash exec.sh client

docker run -it sd2425-lab1-xxxxx-yyyyy bash exec.sh server
```

Again, optionally you can attribute a hostname and name with options -h and --name