Distributed Systems

2021/22

Lab 1

Nuno Preguiça, João Leitão, Dina Borrego, João Vilalonga

GOAL

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

- Use maven to compile, assemble and create a docker image
- Understand the basics of docker
- Use multicast to discover servers in Java

GOAL

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

- **Use maven to compile, assemble and create a docker image**
- Understand the basics of docker
- Use multicast to discover 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 use the option to **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>sd2122</groupId>
        <artifactId>sd2122-aula1</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>3.10.1</version>
            <configuration>
                <source>17</source>
                <target>17</target>
            </configuration>
        </plugin>
```

Allow 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 **jar** file with all code.

POM.XML – CONFIGURATION FILE (CONT)

Creates a docker image. The name uses the property defined before.

```
<plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>0.39.1</version>
    <executions>
        <execution>
            <id>build-dockerimage</id>
            <phase>install</phase>
            <goals>
                <goal>build</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <images>
            <image>
                <name>sd2122-aula1-${authors}</name>
                <build>
            <dockerFile>${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, assemble and create a docker image
- **Understand the basics of docker**
- Use multicast to discover servers in Java

DOCKER

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

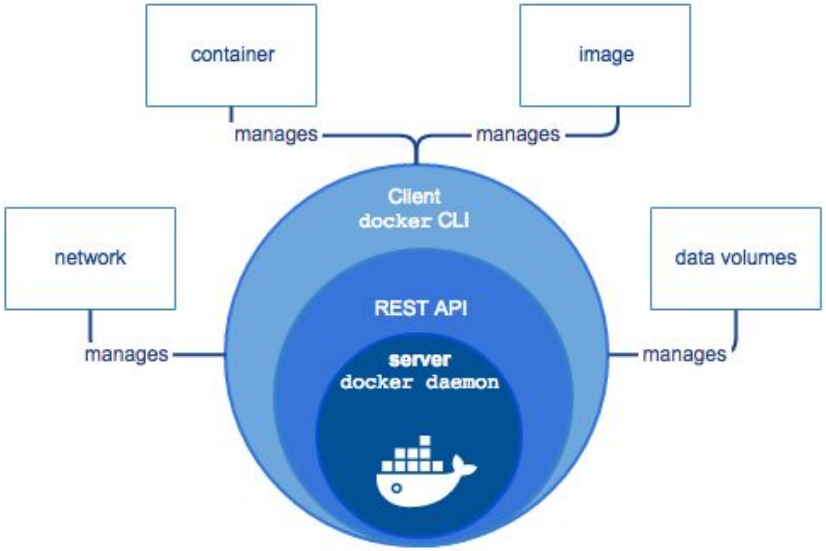
Docker allows multiple services, with multiple instances, to be deployed in the same host computer (cheaply).

A **container** includes all software necessary to run the application and each container executes **isolated** from the other containers, as if **each** was running in a dedicated machine, with its own separate storage and networking stack.

In this course will be useful to run and test exercise solutions and, especially, the two projects.

DOCKER ENGINE

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

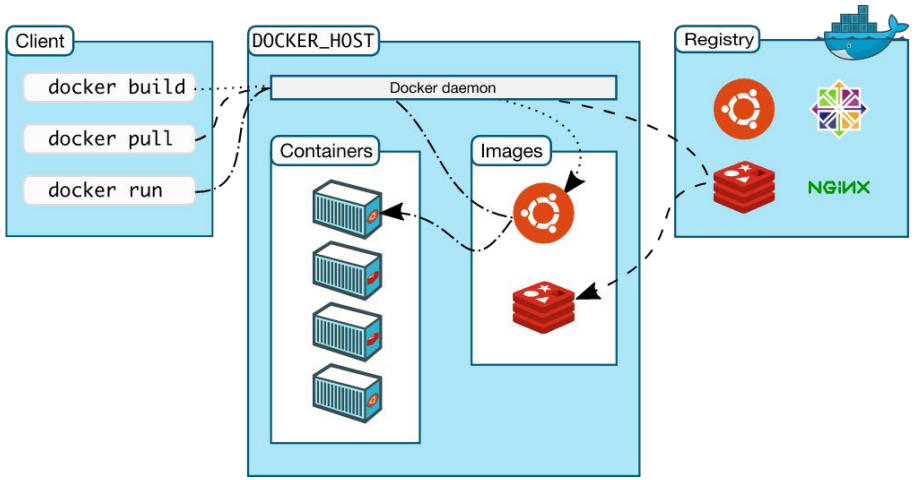


Docker engine (2)

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

FROM openjdk:17

 # FROM defines the image that will be extended

 # openjdk:17 is an image with the open jdk v.17 software

WORKDIR /home/sd

 # WORKDIR defines the directory to be used in the following instructions

COPY target/*jar-with-dependencies.jar sd2122.jar

 # COPY copies the jar to the docker image

CMD ["java", "-cp", "/home/sd/sd2122.jar", "sd2122.aula1.Discovery"]

 # CMD defines the program that will run by default

DOCKERFILE

```
FROM openjdk:17

WORKDIR /home/sd

COPY
target/*jar-with-dependencies.jar
sd2122.jar

CMD ["java", "-cp",
"/home/sd/sd2122.jar",
"sd2122.aula1.Discovery"]
```

FROM defines the image that will be extended

- openjdk:17 is an image with the open jdk v.17 software

WORKDIR defines the directory to be used in the following instructions

COPY copies a file to the docker image

CMD defines the program that will run by default

CREATING A CONTAINER IMAGE

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

mvn docker:build

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

docker build -t name dir_of_dockerfile

docker build -t sd2122-aula1-xxxxx-yyyyy .

-t is used to define the name of the image

. implies the command needs to be run in the directory where the Dockerfile is located

DOCKER: USEFUL COMMANDS

Docker run command:

docker run [params] imagename [cmd]

Start an image and run the default command:

docker run sd2122-aula1-xxxxx-yyyyy

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

docker run -it sd2122-aula1-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 sd2122-aula1-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 sd2122-aula1-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 docker images.

DOCKER: MORE USEFUL COMMANDS (2)

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

Lists docker images.

docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Pulls an image from a registry.

docker push [OPTIONS] NAME[:TAG]

Push an image to a registry.

GOAL

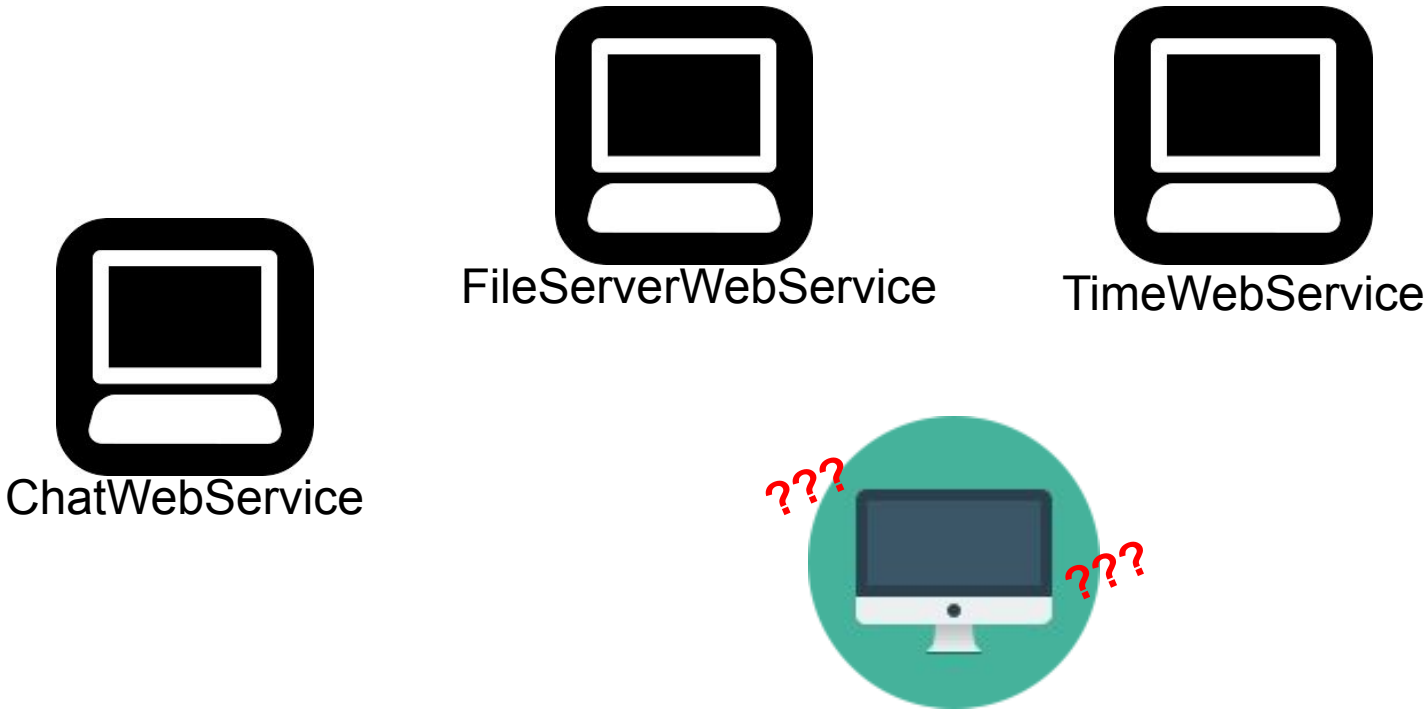
In the end of this lab you should be able to:

- Use maven to compile, assemble and create a docker image
- Understand the basics of docker
- **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?

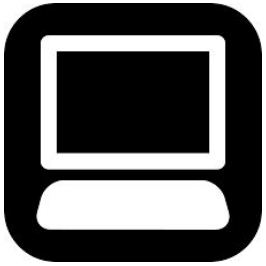


How to perform service discovery?

One solution is to use **IP Multicast**

(There are two flavors)

ChatWebService



FileServerWebService

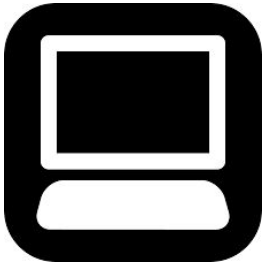
TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

1st Alternative: Server Initiated

ChatWebService



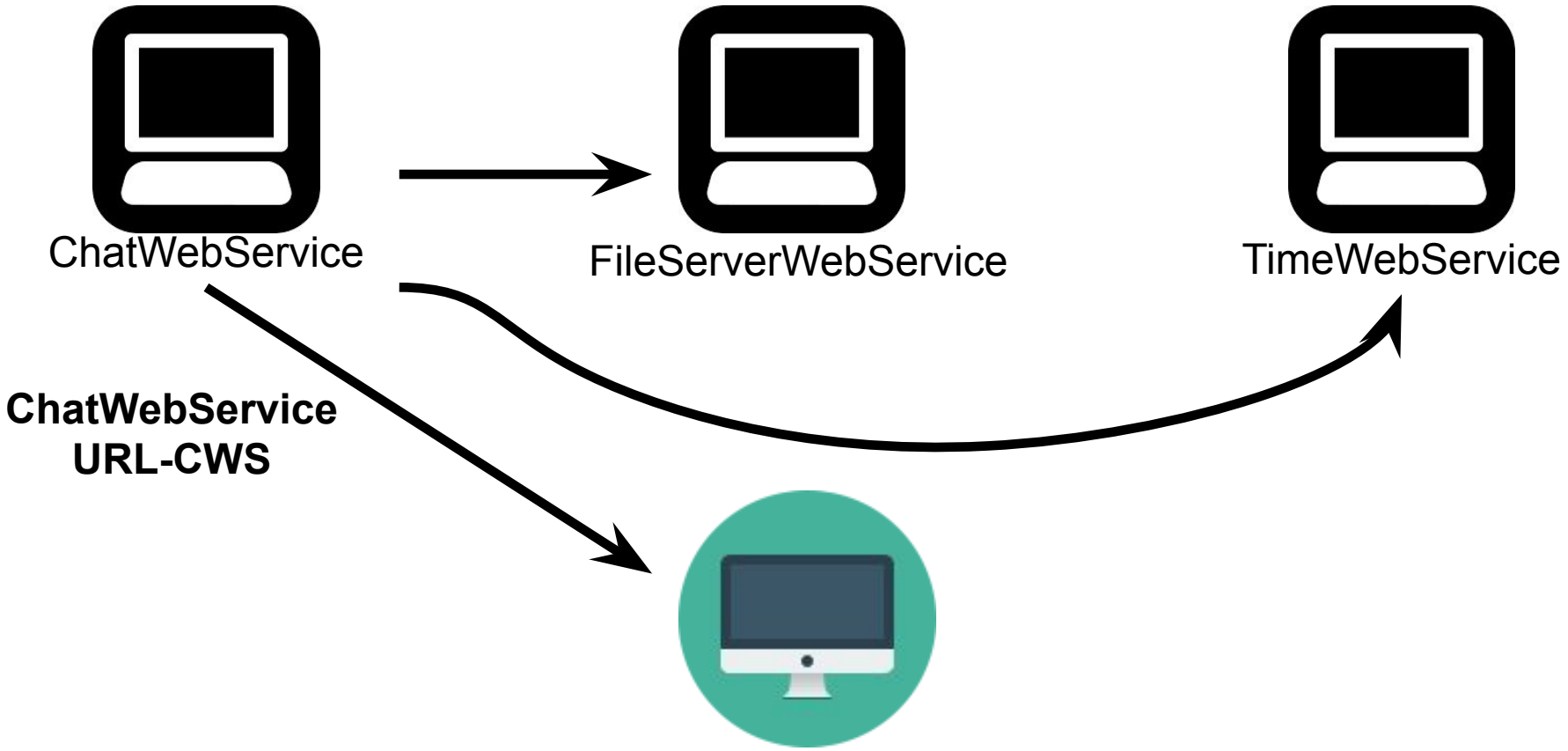
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

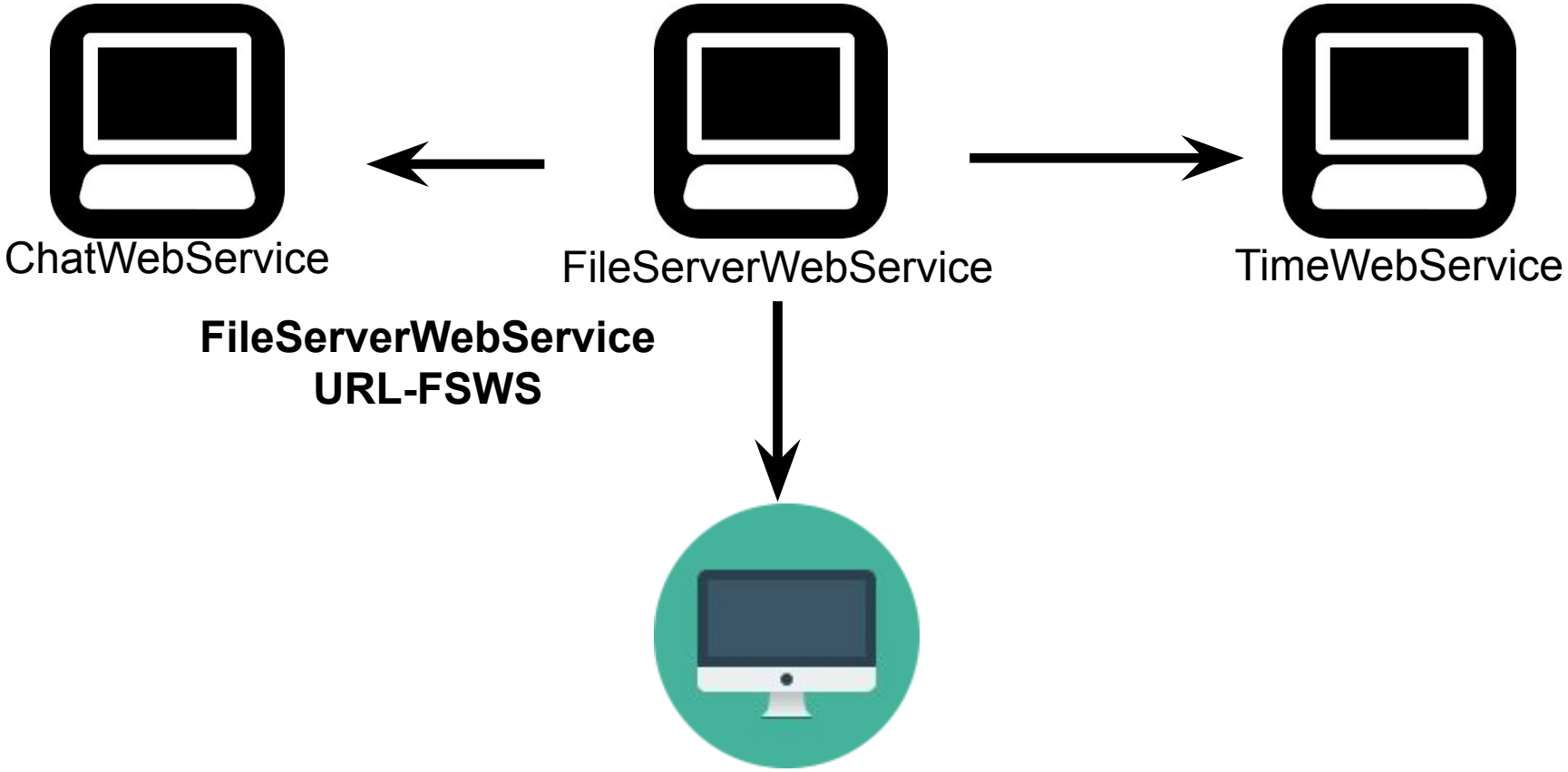
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

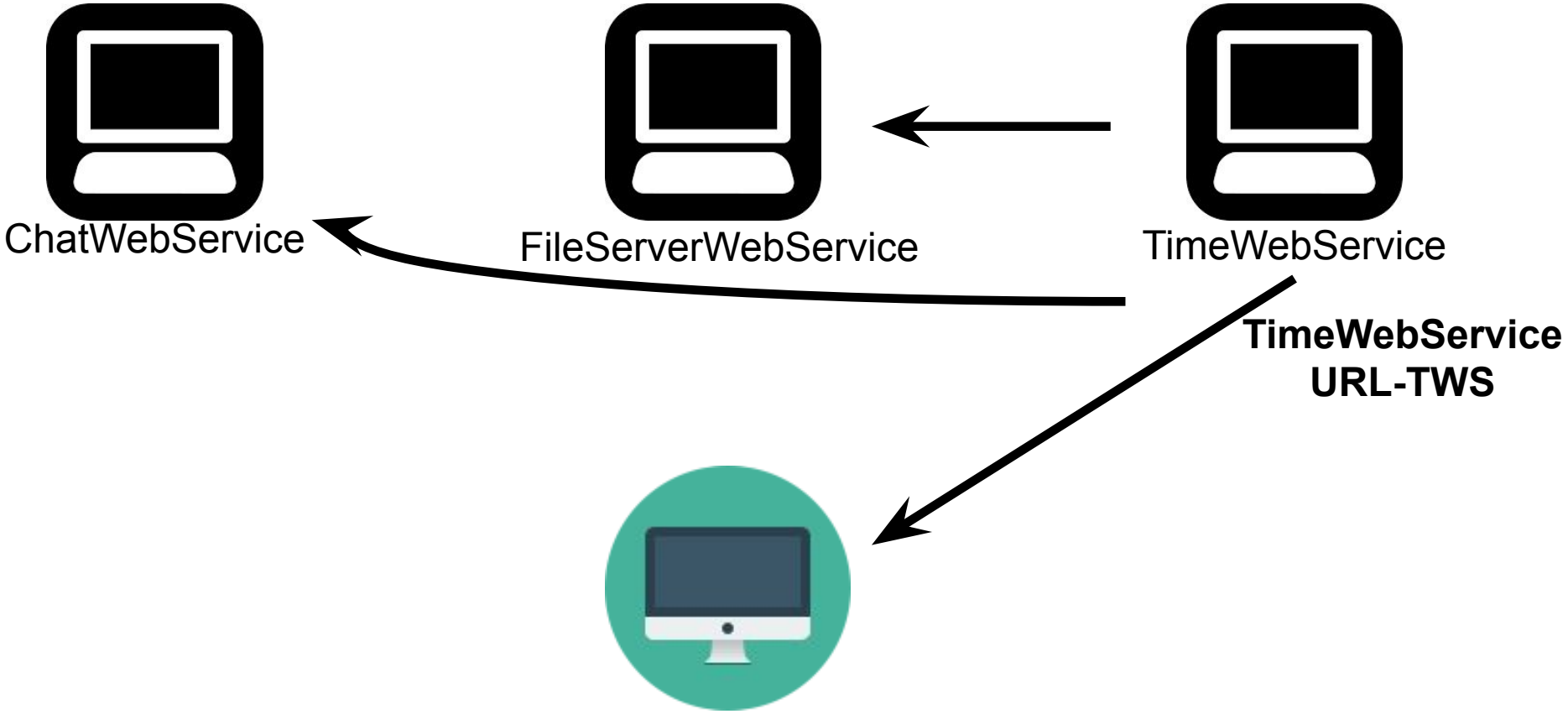
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

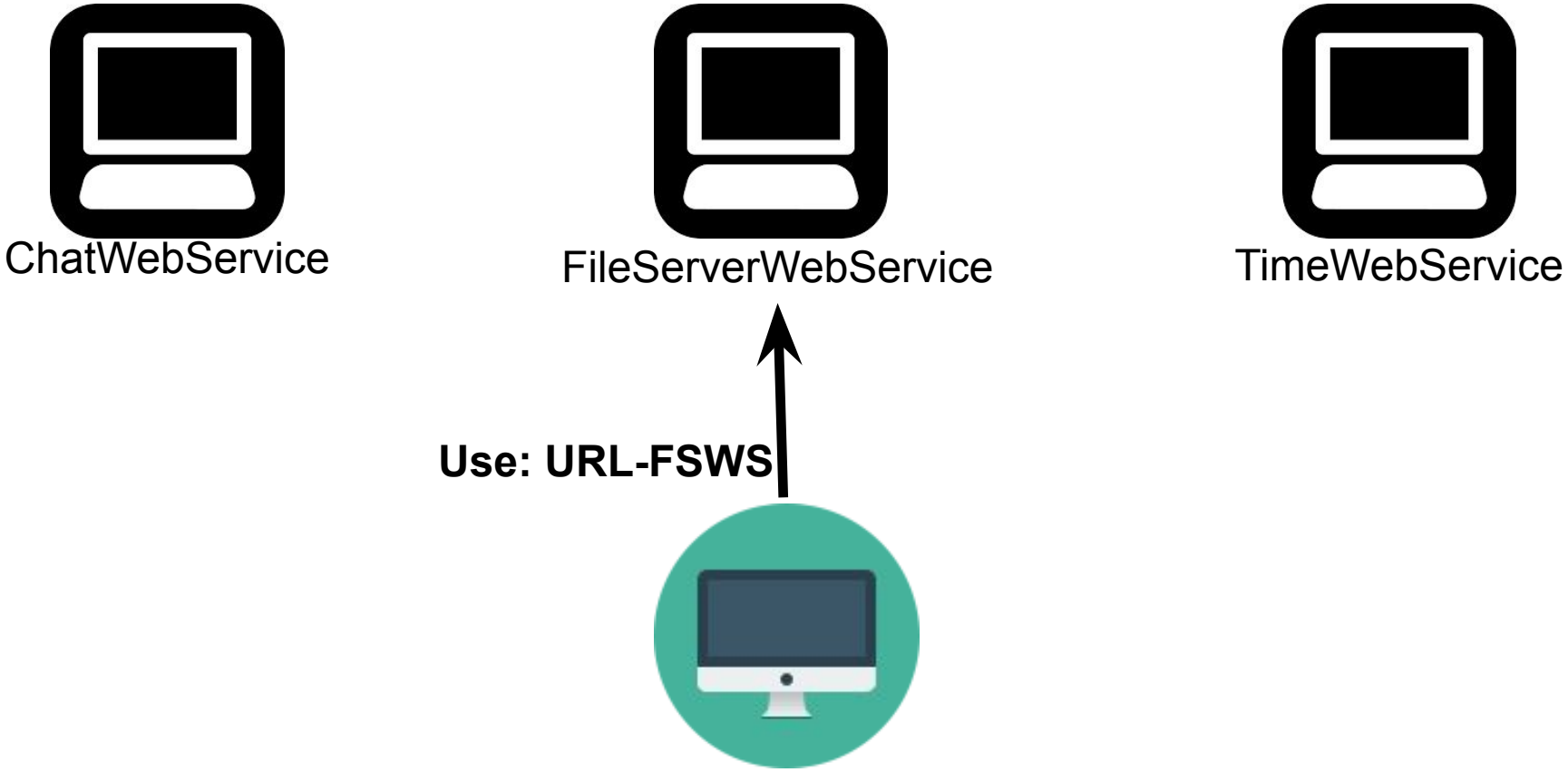
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

1st Alternative: Server Initiated

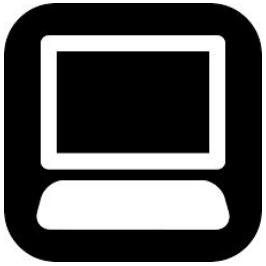


How to perform service discovery?

One solution is to use **IP Multicast**

2nd Alternative: Client Initiated

ChatWebService



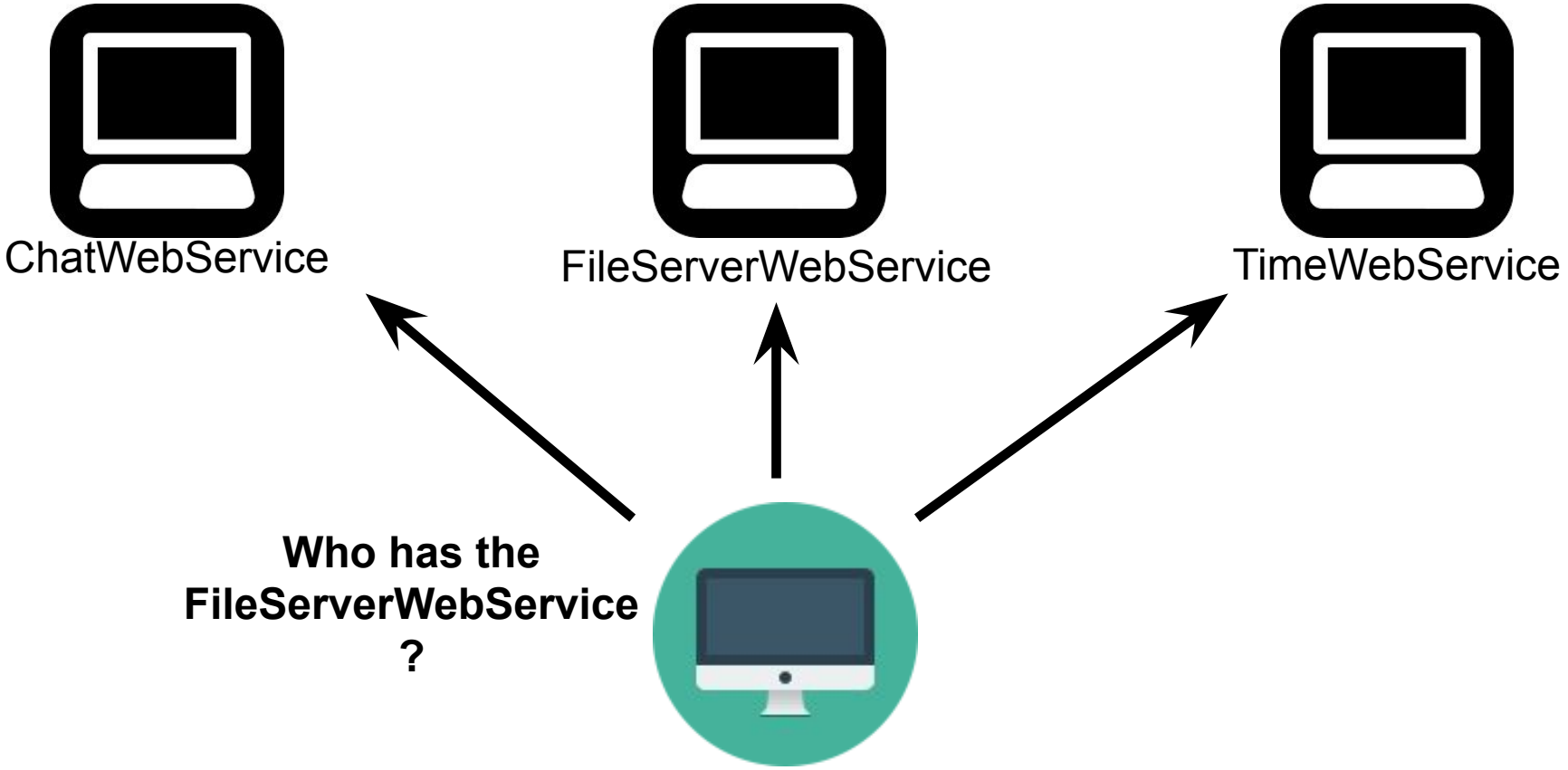
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

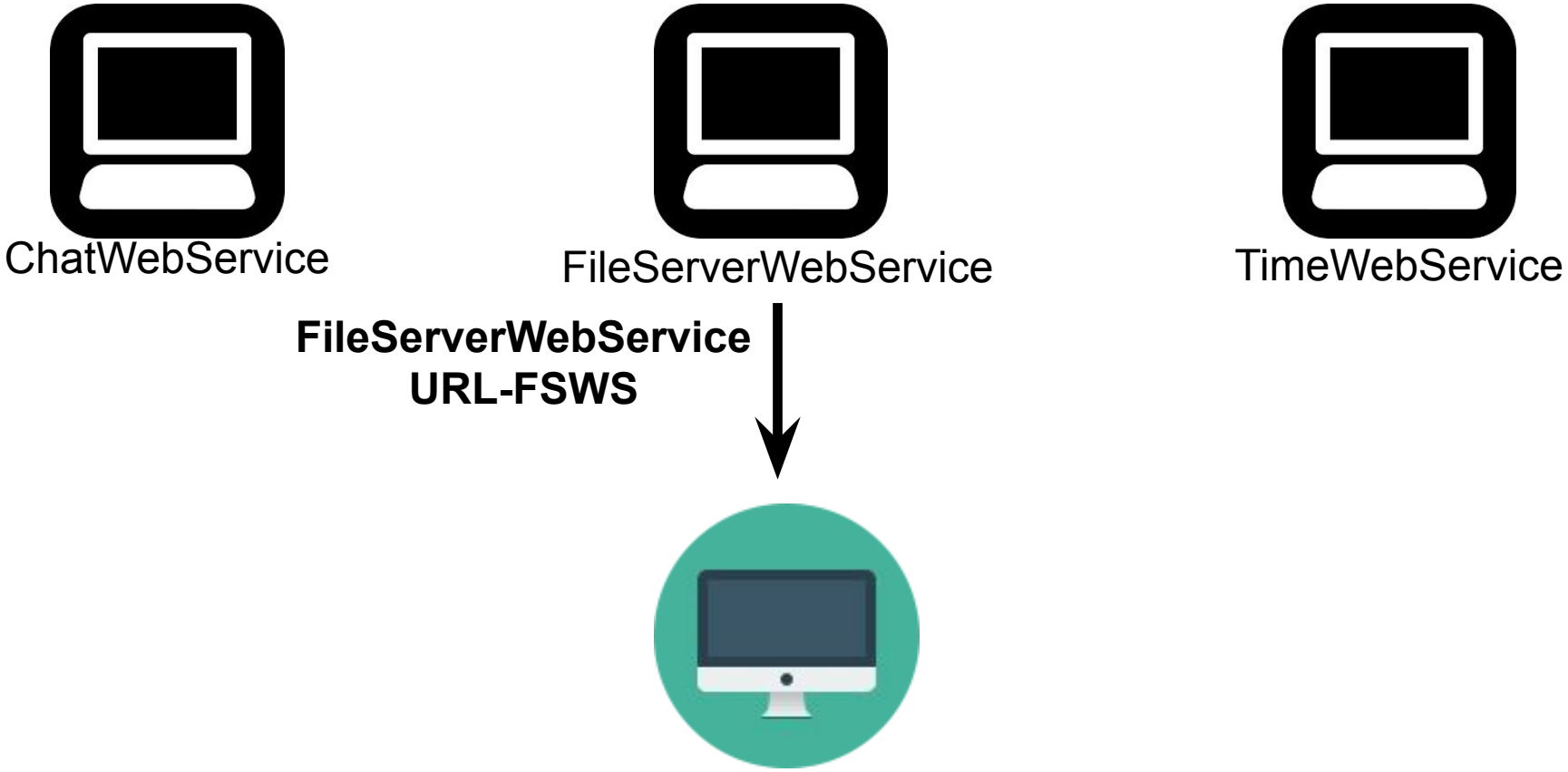
2nd Alternative: Client Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

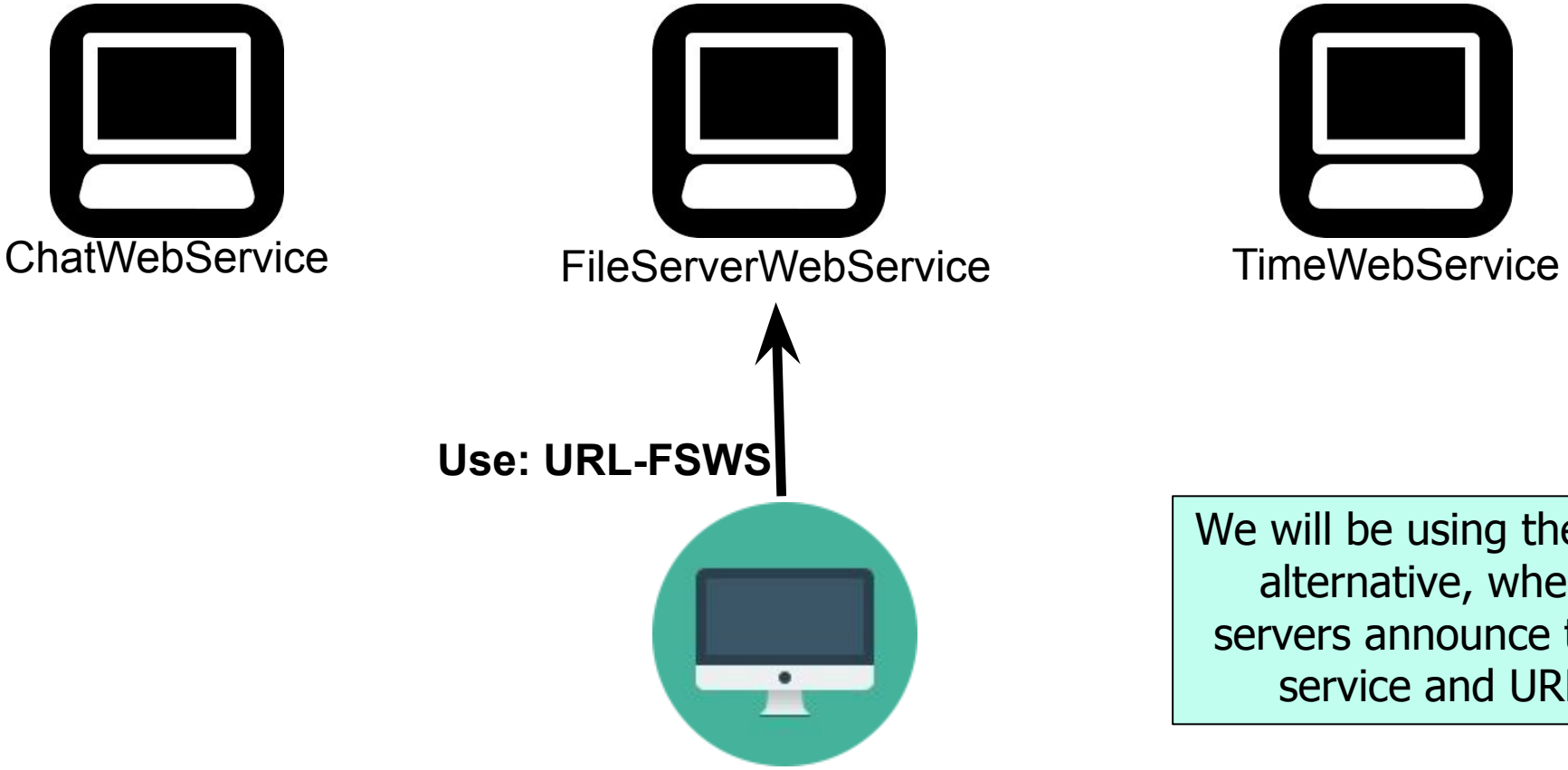
2nd Alternative: Client Initiated



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 : ANNOUNCEMENT

```
var pktBytes = String.format("%s%s%s", serviceName, DELIMITER, serviceURI).getBytes();

var pkt = new DatagramPacket(pktBytes, pktBytes.length, DISCOVERY_ADDR);
// start thread to send periodic announcements
new Thread(() -> {
    try (var ds = new DatagramSocket()) {
        for (;;) {
            try {
                Thread.sleep(DISCOVERY_PERIOD);
                ds.send(pkt);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}).start();
```

SERVICE DISCOVERY : ANNOUNCEMENT

```
var pktBytes = String.format("%s%s%s", serviceName, DELIMITER, serviceURI).getBytes();

var pkt = new DatagramPacket(pktBytes, pktBytes.length, DISCOVERY_ADDR);
// start thread to send periodic announcements
new Thread(() -> {
    try (var ds = new DatagramSocket()) {
        for (;;) {
            try {
                Thread.sleep(DISCOVERY_PERIOD);
                ds.send(pkt);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}).start();
```

Create a socket to send announcements.

The try construct automatically closes the socket when the code exits.

```
try( ... ) {
}
```

SERVICE DISCOVERY : ANNOUNCEMENT

```
var pktBytes = String.format("%s%s%s", serviceName, DELIMITER, serviceURI).getBytes();

var pkt = new DatagramPacket(pktBytes, pktBytes.length, DISCOVERY_ADDR);
// start thread to send periodic announcements
new Thread(() -> {
    try (var ds = new DatagramSocket()) {
        for (;;) {
            try {
                Thread.sleep(DISCOVERY_PERIOD);
                ds.send(pkt);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}).start();
```

Periodically sends the announcement message.

SERVICE DISCOVERY : DISCOVERY

```
final int MAX_DATAGRAM_SIZE = 1024;
var pkt = new DatagramPacket(new byte[MAX_DATAGRAM_SIZE], MAX_DATAGRAM_SIZE);


new Thread(() -> {
   try (var ms = new MulticastSocket(DISCOVERY_ADDR.getPort())) {
      joinGroupInAllInterfaces(ms);
      for(;;) {
         ...
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
}).start();
```

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

SERVICE DISCOVERY : DISCOVERY

```
for(;;) {
   try {
      pkt.setLength(MAX_DATAGRAM_SIZE);
      ms.receive(pkt);

      var msg = new String(pkt.getData(), 0, pkt.getLength());
      var tokens = msg.split(DELIMITER);
      System.out.printf( "FROM %s (%s) : %s\n",
        pkt.getAddress().getCanonicalHostName(), pkt.getAddress().getHostAddress(),
        msg);

      if (tokens.length == 2) {
         //TODO: complete by recording the received information from the other node.
      }
   } catch (IOException e) {
      try {
         Thread.sleep(DISCOVERY_PERIOD);
      } catch (InterruptedException e1) {
         // do nothing
   }
   Log.finest("Still listening...");
}
```

Receive and process message.

EXERCISE

1. Run multiple container images and verify that each container will receive announcements from its own and other containers.
2. Complete the code to store in memory the information about running services.
    - Suggestion: also store the time of received announcements to know which servers are likely available.

NOTE: this code will be used in your project for discovering servers.