DISTRIBUTED SYSTEMS

Lab 1

Nuno Preguiça, João Leitão, Pedro Fouto, Luís Silva

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

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

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>sd1920</groupId>
        <artifactId>sd1920-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>3.8.0</version>
                        <configuration>
                                <source>1.8</source>
                                <target>1.8</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 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>0.33.0</version>
        <executions>
                <execution>
                        <id>build-dockerimage</id>
                        <phase>install</phase>
                        <goals>
                                <goal>build</goal>
                        </goals>
                </execution>
        </executions>
        <configuration>
                <images>
                        <image>
                                <name>sd1920-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 dockerfile: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

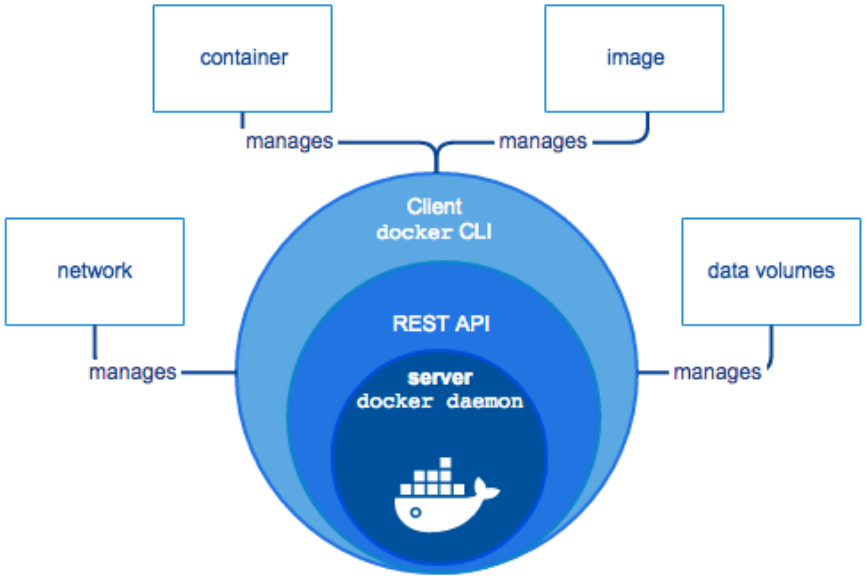
DOCKER

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.

DOCKER ENGINE

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

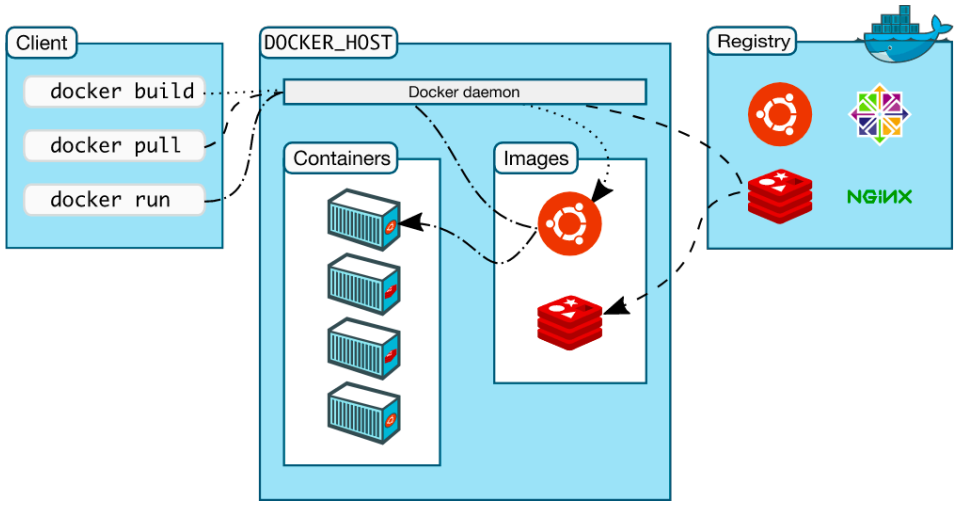


DOCKER ENGINE (2)

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 defines the image that will be extended
# openjdk:8 is an image with the  open jdk v.8 software
FROM openjdk:8

# WORKDIR  defines the directory to be used in the following instructions
WORKDIR /home/sd

# COPY copies the jar to the docker image
COPY target/*jar-with-dependencies.jar sd1920.jar

# CMD defines the program that  will run  by default
CMD ["java", "-cp", "/home/sd/sd1920.jar", "sd1920.aula1.Discovery"]
```

CREATING A CONTAINER IMAGE

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

mvn dockerfile: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 sd1920-aula1-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 sd1920-aula1-xxxxx-yyyyy

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

docker run -it sd1920-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 sd1920-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 sd1920-aula1-xxxxx-yyyyy

DOCKER: MORE USEFUL COMMANDS

`docker ps [OPTIONS]`

Lists containers.

`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

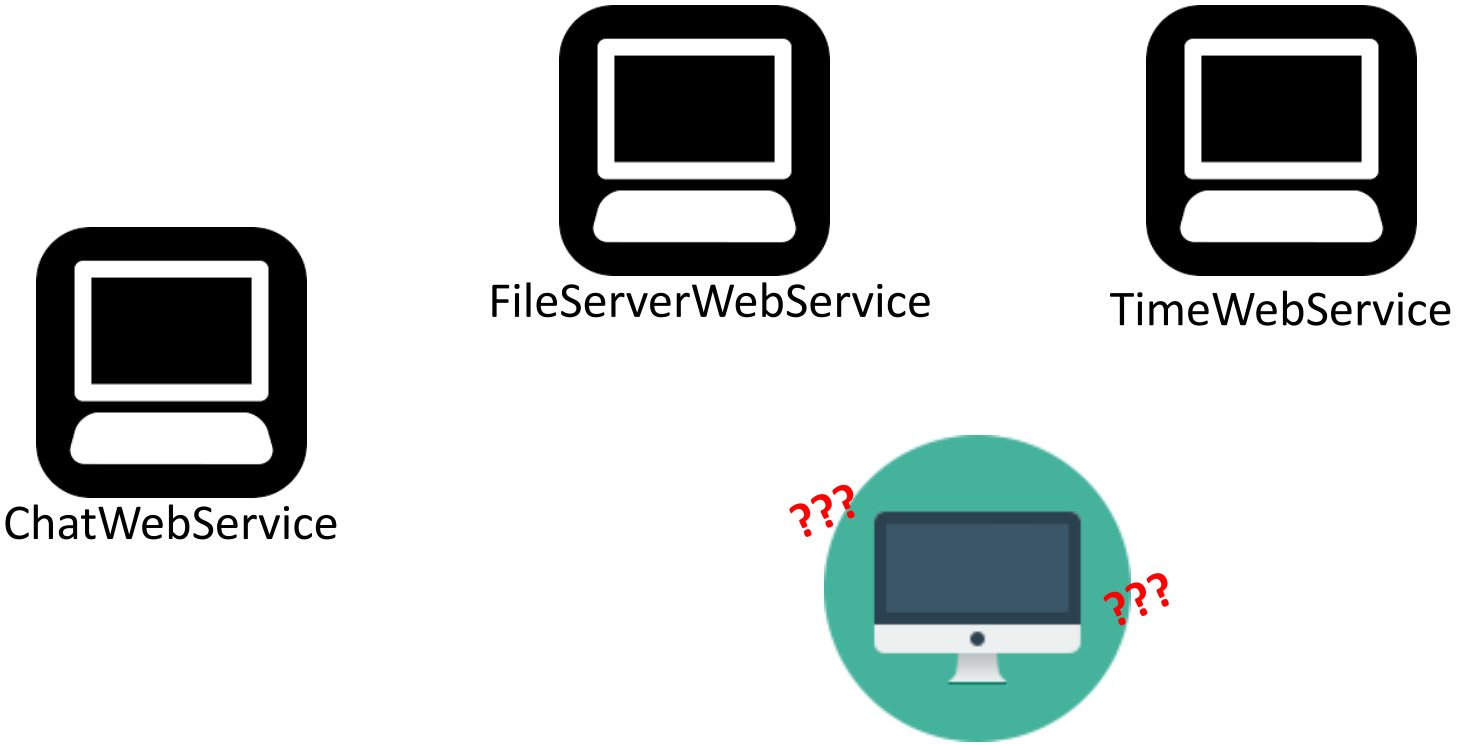
In the end of this lab you should be able to:

- 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

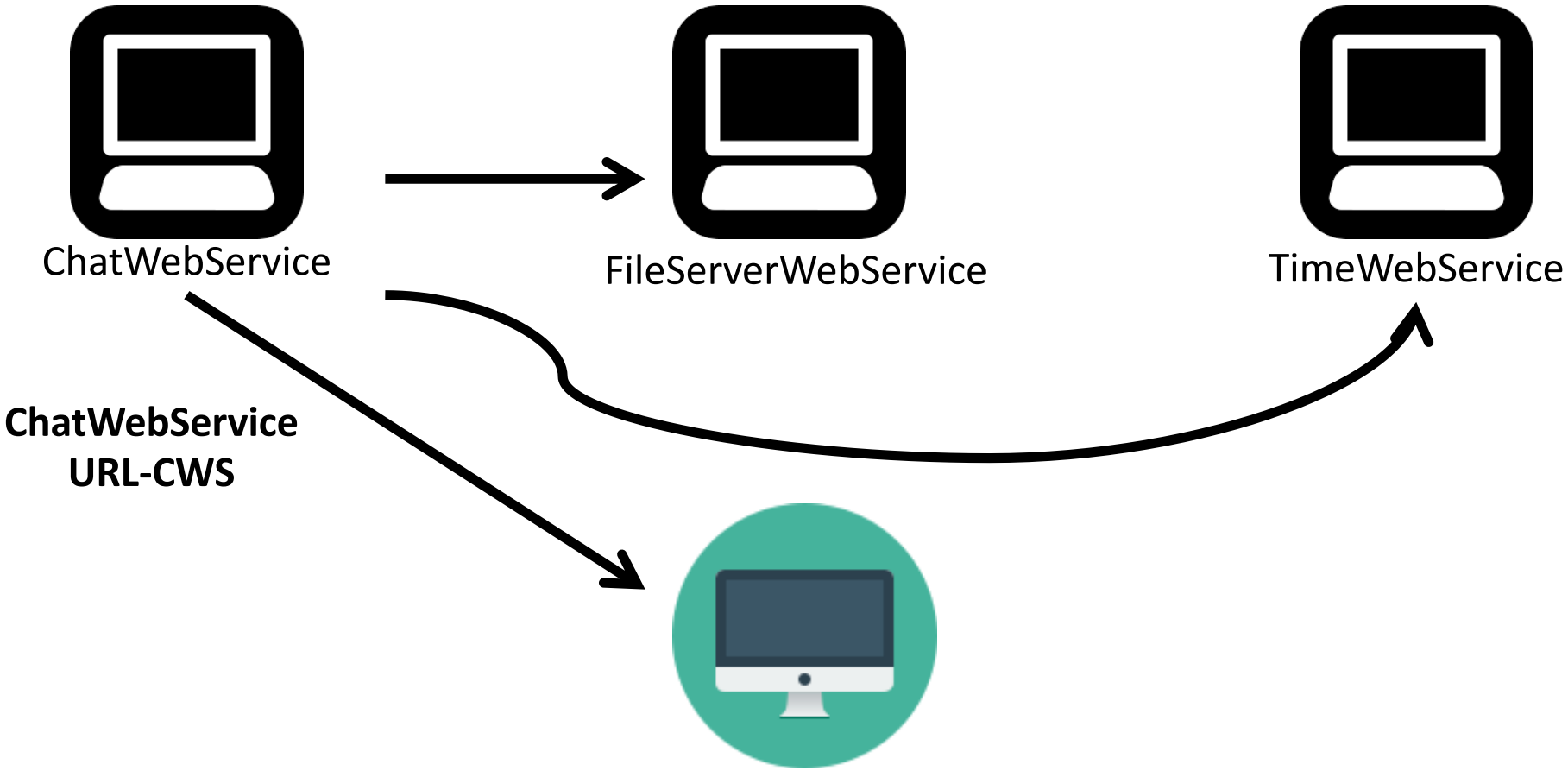
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

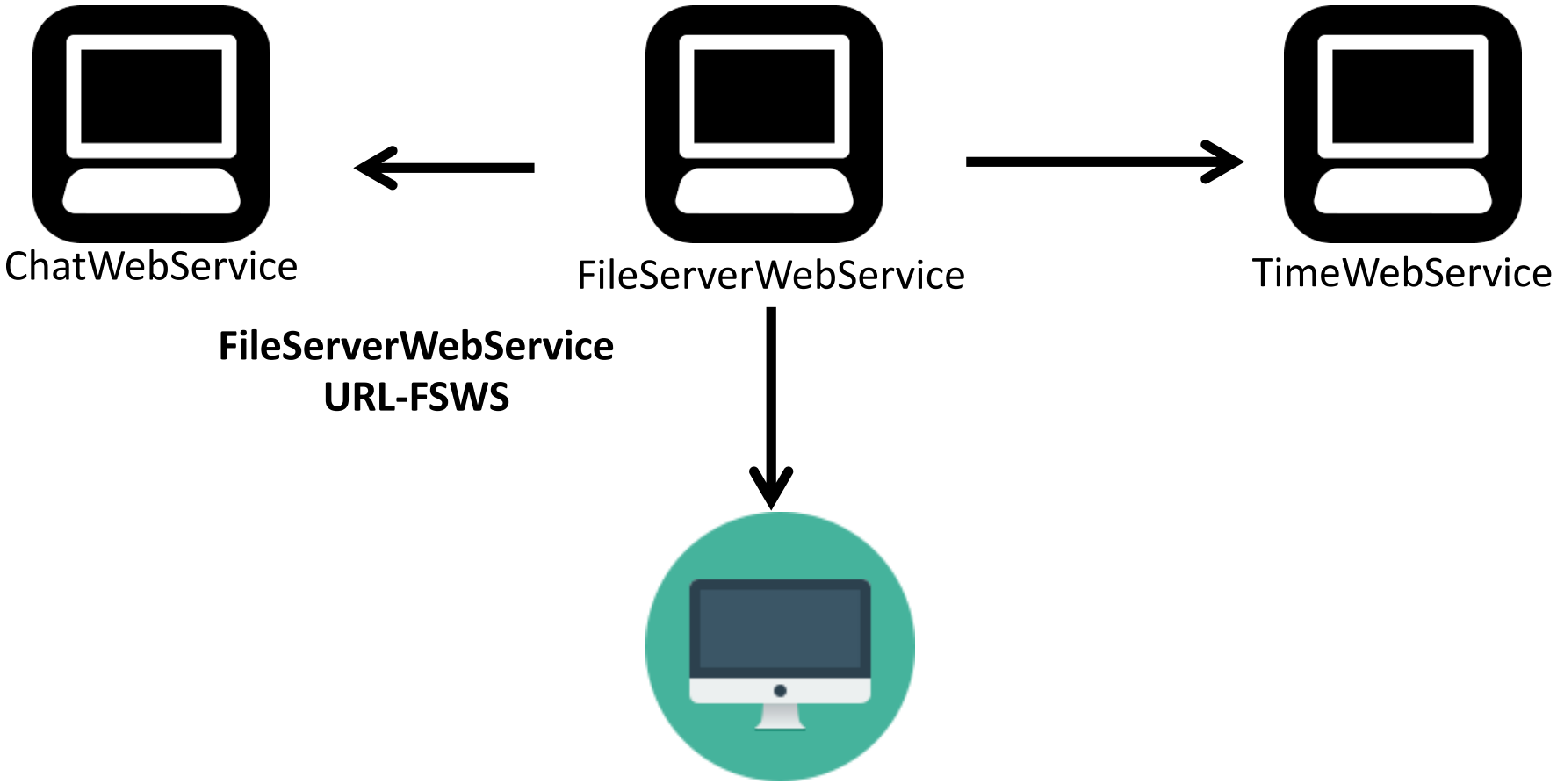
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

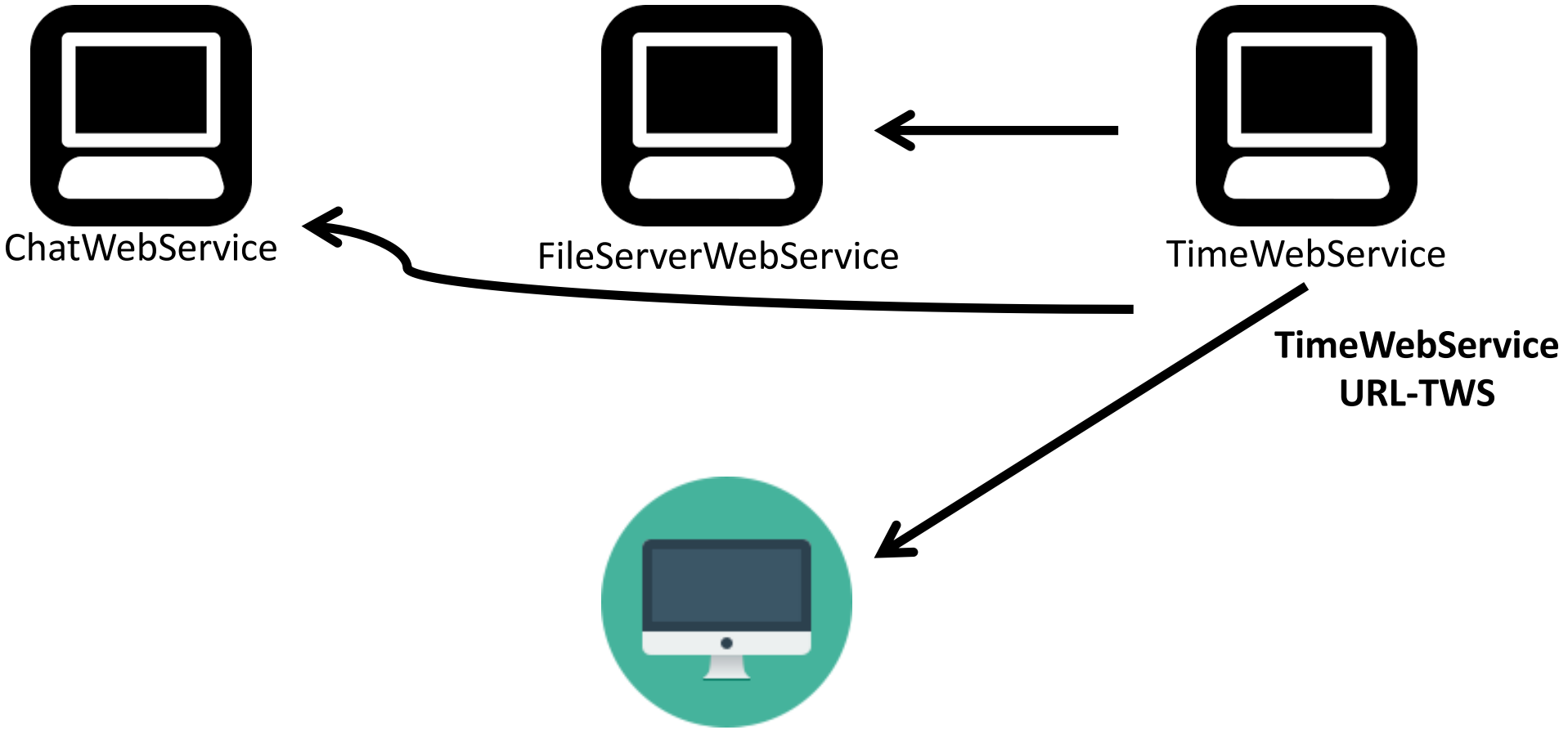
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

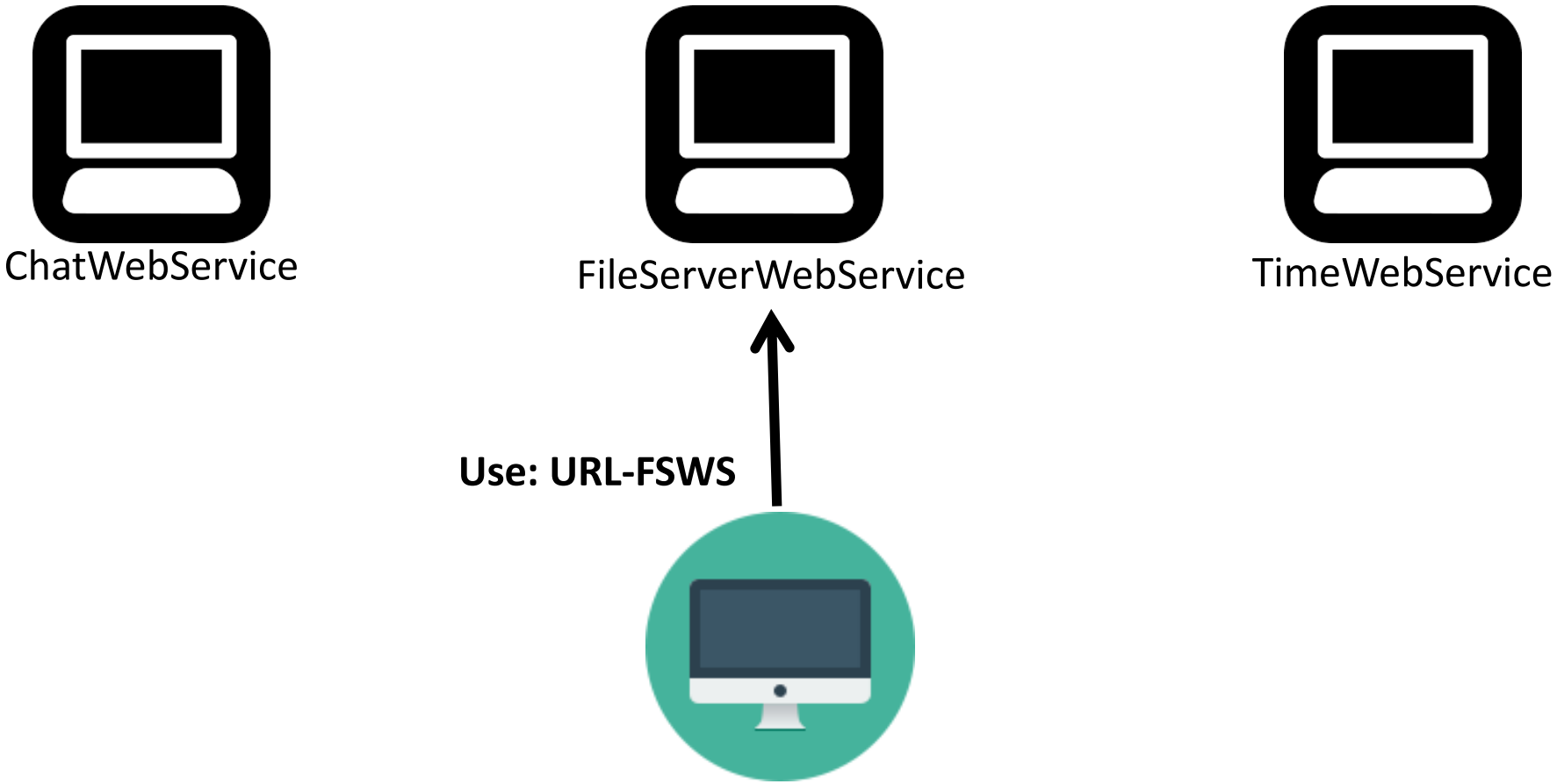
1st Alternative: Server Initiated



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

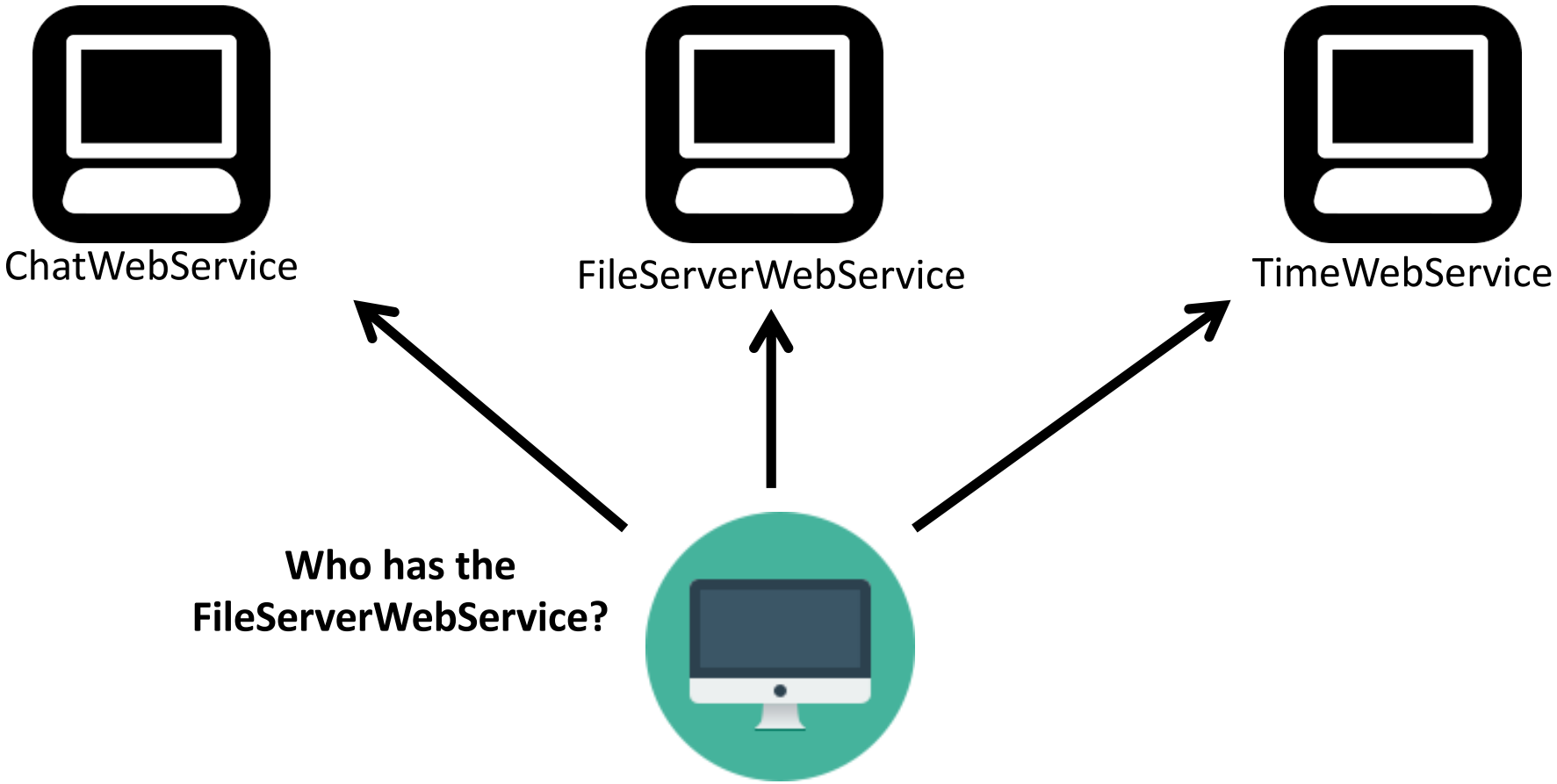
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

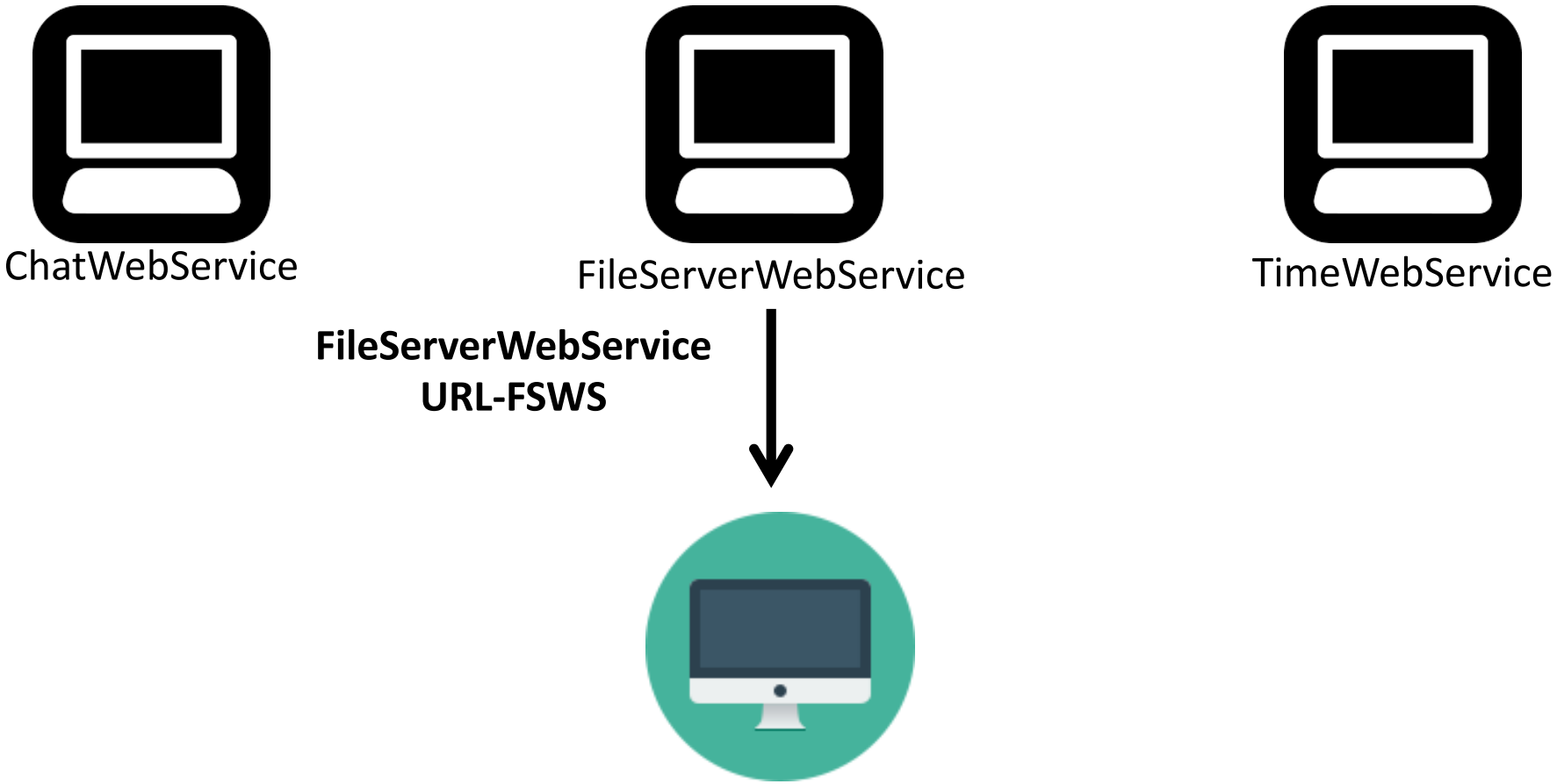
2nd Alternative: Client Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

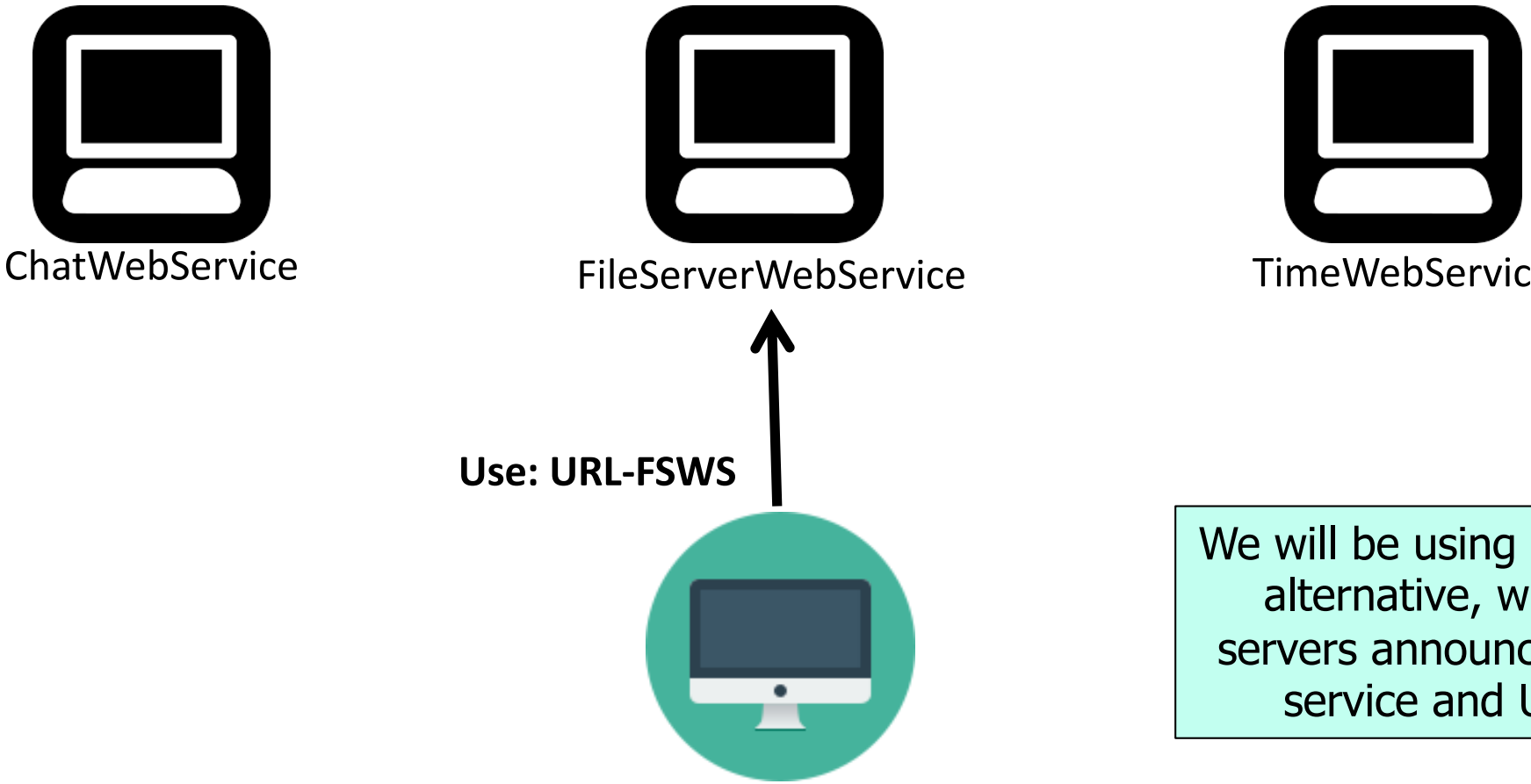
2nd Alternative: Client Initiated



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

```
byte[] annBytes = String.format("%s%s%s", serviceName, DELIMITER,
serviceURI).getBytes();
DatagramPacket announcePkt = new DatagramPacket(annBytes, annBytes.length, addr);

try {
    MulticastSocket ms = new MulticastSocket( addr.getPort());
    ms.joinGroup(addr.getAddress());

    // start thread to send periodic announcements
    new Thread(() -> {
        for (;;) {
            try {
                ms.send(announcePkt);
                Thread.sleep(DISCOVERY_PERIOD);
            } catch (Exception e) {
                e.printStackTrace();
                // do nothing
            }
        }
    }).start();
```

SERVICE DISCOVERY

```
byte[] annBytes = String.format("%s%s%s", serviceName, DELIMITER,
serviceURI).getBytes();
DatagramPacket announcePkt = new DatagramPacket(annBytes, annBytes.length, addr);

try {
    MulticastSocket ms = new MulticastSocket( addr.getPort());
    ms.joinGroup(addr.getAddress());

    // start thread to send periodic announcements
    new Thread(() -> {
        for (;;) {
            try {
                ms.send(announcePkt);
                Thread.sleep(DISCOVERY_PERIOD);
            } catch (Exception e) {
                e.printStackTrace();
                // do nothing
            }
        }
    }).start();
```

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

SERVICE DISCOVERY

```
byte[] annBytes = String.format("%s%s%s", serviceName, DELIMITER,
serviceURI).getBytes();
DatagramPacket announcePkt = new DatagramPacket(annBytes, annBytes.length, addr);

try {
    MulticastSocket ms = new MulticastSocket( addr.getPort());
    ms.joinGroup(addr.getAddress());

    // start thread to send periodic announcements
    new Thread(() -> {
        for (;;) {
            try {
                ms.send(announcePkt);
                Thread.sleep(DISCOVERY_PERIOD);
            } catch (Exception e) {
                e.printStackTrace();
                // do nothing
            }
        }
    }).start();
```

Periodically send the announcement message.

SERVICE DISCOVERY (2)

```
// start thread to collect announcements
new Thread(() -> {
    DatagramPacket pkt = new DatagramPacket(new byte[1024], 1024);
    for (;;) {
        try {
            pkt.setLength(1024);
            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().getCanonicalHostName(),
                                   pkt.getAddress().getHostAddress(), msg);
                //TODO: to complete by recording the received information
            }
        } catch (IOException e) {
            // do nothing
        }
    }
}).start();
```

SERVICE DISCOVERY (2)

```
// start thread to collect announcements
new Thread(() -> {
    DatagramPacket pkt = new DatagramPacket(new byte[1024], 1024);
    for (;;) {
        try {
            pkt.setLength(1024);
            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().getCanonicalHostName(),
                                   pkt.getAddress().getHostAddress(), msg);
                //TODO: to complete by recording the received information
            }
        } catch (IOException e) {
            // do nothing
        }
    }
}).start();
```

Receive and process message.

EXERCISE

1. Run multiple container images and verify that each container will receive announcement from its own and other containers.
2. Complete the code to record information about running services. Suggestion: store the time of received announcement to know which servers are currently reachable.

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