GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Assessable Task 2nd Term

Computer Systems 23/24

Desarrollo de Aplicaciones Web

Aarón Martín Bermejo
Francisco Lifante
Álvaro Maceda

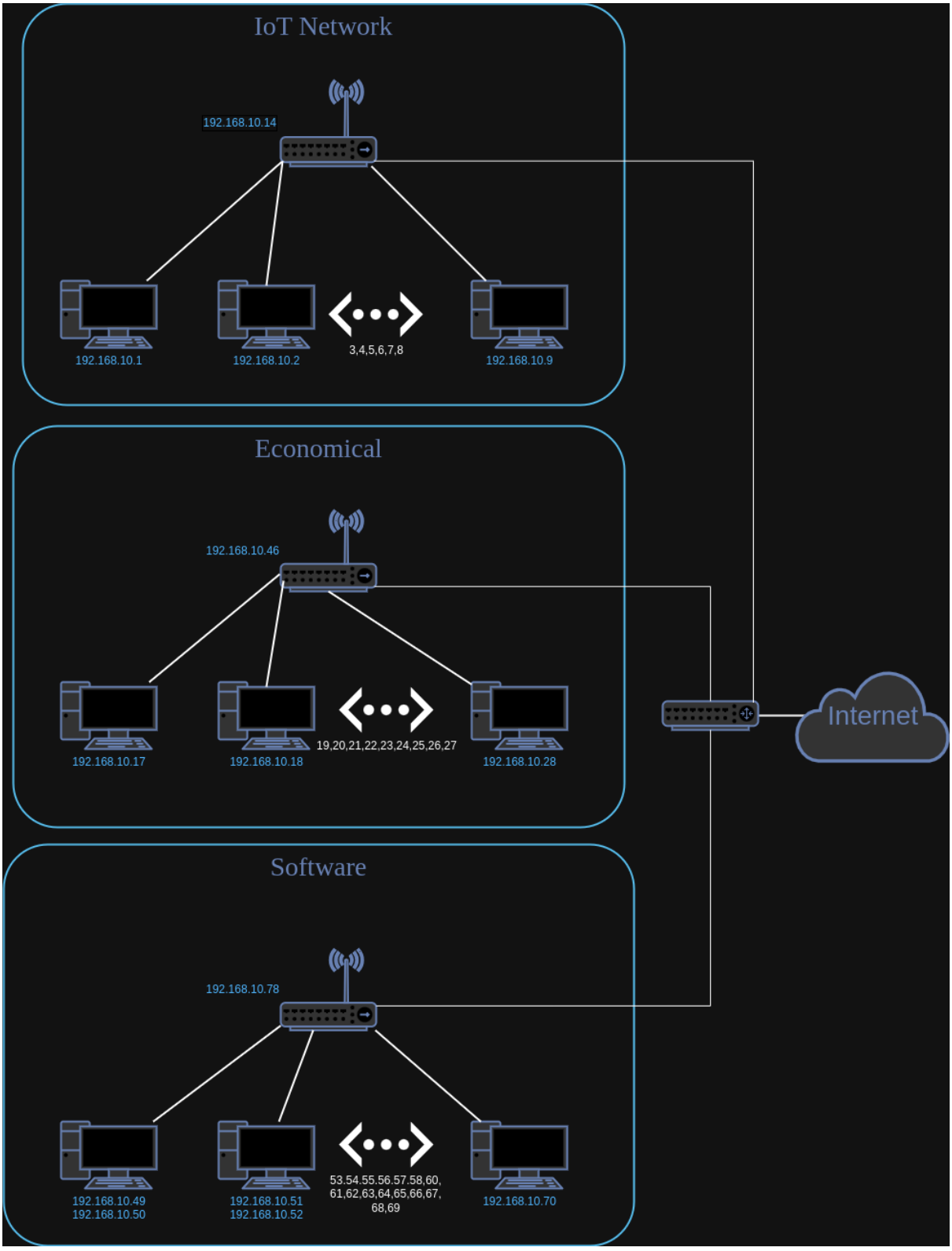
Cicles Formatius

License

TABLE OF CONTENTS

Exercise 1

Section a



Section b

Subnet	Network Address	Broadcast Address	Mask	Number of usable hosts
IoT	192.168.10.0	192.168.10.15	28	14
Economical	192.168.10.16	192.168.10.47	27	30
Software	192.168.10.48	192.168.10.79	27	30

For the economical and software networks we will need at least 16 IPs)will be explained after), which fits in networks with a 27 mask:

1111 1111 1110 0000

$2^5 = 32$

Because with a 28 mask we will only have available 14 hosts available (16 – network address – broadcast address):

1111 1111 1111 0000

$2^4 = 16$

We could create the three networks with a 27 fixed mask, but to make it more efficient we're setting up the IoT with 14 hosts because we will need at least 9 IPs plus the router. So 10 IPs the closest mask that we can get is 28 because that mask gives us 14 available hosts (16 minus network address and broadcast address).

As previously said, for the economical network we will need 15 IPs plus the router (because the router should also have an IP to communicate with it) so the 28 mask won't be enough. We'll need a 27 mask so we have 30 available hosts (32 minus network address and broadcast).

About software network, as we will need 22 IPs for software (10 * 2 for devs and 2 more for sysadmins) plus the router closest mask is 27 with 30 available hosts (32 minus network address and broadcast).

Network IoT

Device	IP
Device 1	192.168.10.1
Device 2	192.168.10.2
…	…
Device 9	192.168.10.9

Router	192.168.10.14
Empty IPs range	192.168.10.10 – 192.168.10.13

Network Economical

Device	**IP**
Device 1	192.168.10.17
Device 2	192.168.10.18
…	
Device 12	192.168.10.28
Router	192.168.10.46
Empty IPs range	192.168.10.29 – 192.168.10.45

Network Software

Device	**IP**
Device 1	192.168.10.49
Device 2	192.168.10.50
…	
Device 22	192.168.10.70
Router	192.168.10.78
Empty IPs range	192.168.10.71 – 192.168.10.77

Usally the router IP is the first available in the network, but it's just a convention or something easy because usually the router is the first device connected in the network.

Exercise 2

Section a: Launch the Redis server

```
docker run --rm \
    --name redis-server \
    -p 8001:8001 \
    redis/redis-stack:latest
```

Section b: Run the Flask app

For the first section you must run the command:

```
docker run --rm -ti \
    --name flask-server \
    -p 5000:5000 \
    --mount type=bind,source="$(pwd)",destination=/python \
    python:3.8-slim \
    /bin/bash
```

For the rest, just use the commands given, changing the directory with `cd`.

Section c: Create Flask image

1) Dockerfile:

```
# Use a Python base image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt /

# Install dependencies
RUN pip install --no-cache-dir -r /requirements.txt

# Set environment variables
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_ENV=development
ENV REDIS_SERVER=FILL-LATER

# Define the command to run the Flask application
CMD ["flask", "run", "--debug"]
```

2) Build the image:

```
docker build -t flask-app .
```

3) Run the container from the base directory:

```
docker run --rm -d \
    --name my-app-container \
    -p 5000:5000 \
    --mount type=bind,source="$(pwd)/app",destination=/app \
    flask-app
```

Section d: Launch the two containers together

You need to create a private network:

```
docker network create my-app-network
```

1) Launch the Redis container in that network. It's the same command as section a, but includes the network:

```
docker run --rm -d \
    --network my-app-network \
    --name redis-server \
    -p 8001:8001 \
    redis/redis-stack:latest
```

2) Modify the Dockerfile. The name of the host in the private network is the name of the container, so you must use the `--name` parameter from the previous command (`redis-server` in this example):

```
# Use a Python base image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt /

# Install dependencies
RUN pip install --no-cache-dir -r /requirements.txt

# Set environment variables
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_ENV=development
ENV REDIS_SERVER=redis-server

# Define the command to run the Flask application
CMD ["flask", "run", "--debug"]
```

3) Build the image with the same command used in section c:

```
docker build -t flask-app .
```

4) Launch the server with the same command used in section c, but including the network:

```
docker run --rm -d \
    --network my-app-network  \
    --name my-app-container \
    -p 5000:5000 \
    --mount type=bind,source="$(pwd)/app",destination=/app \
    flask-app
```