Each partial is worth 10 points and consists of 17 questions. Each question poses 4 alternatives and has only one correct answer. Once the worst answer is discarded, each correct answer earns 10/16 points, and each error deducts 10/48 points. You must answer on the answer sheet.

Partial 1

1 *Let us consider the following pair of Node.js programs that use ØMQ:*

```
// Program: sender.js
const zmq = require("zeromq")
const producer = zmq.socket("push")
let count = 0
producer.bind("tcp://*:8888", (err) => {
  if (err) throw err
  setInterval(function() {
    producer.send("msg# " + count++)
  }, 1000)
})
```

```
// Program: receiver.js
const zmq = require("zeromq")
const consumer = zmq.socket("pull")
consumer.connect("tcp://127.0.0.1:8888")
consumer.on("message", function(msg) {
  console.log("received: " + msg)
})
```

All instances of those programs run **in the same computer***, and we try to start as many processes of each type as possible. What is the maximum number of processes that may be started without generating any error?*

- **a** There is no limit.
- **b** A single sender and many receivers.
- **c** A single sender and a single receiver.
- **d** Many senders and a single receiver.

2 *In Unit 1, we have seen that software services (e.g., those deployed by companies in internal labs with clusters of computers) have migrated to the cloud. An important reason for that migration is…*

- **a** Internal labs are prone to failures.
- **b** Internal labs have many more security vulnerabilities than the data centres managed by cloud providers.
- **c** Internal labs become expensive and difficult to administer, while cloud services follow a pay-per-use model and may provide a self-administered environment.
- **d** Cloud services follow a cooperative computing model that scales trivially.

3 *Among other things, JavaScript has been chosen as the programming language to be discussed in Unit 2 because …*

- **a** It supports multi-threaded programming. Multi-threading is a key characteristic to write simple programs and ensure application correctness.
- **b** Wikipedia has been implemented using JavaScript and many of the examples used in Unit 2 and in the labs are inspired in Wikipedia code snippets.
- **c** All choices are true.
- **d** It supports asynchronous programming. Asynchronous programming provides a good base to design and develop scalable distributed applications.

Let us consider the following program in next two questions

```
// Program: example.js
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
const f = (x) => {
   emitter.emit(e1, x);
   return (y) => {return y+x}
}
emitter.on(e1, function(y) {
  console.log(y+"!!")
})
setTimeout(f("First"), 2000)
emitter.emit(e1,"Second")
console.log("End.")
```

4 *What does 'example.js' show on the screen when it runs?*

a End.
Second!!
First!!

b It shows an error since the setTimeout first argument is not a function.

c End.
First!!
Second!!

d First!!
Second!!
End.

5 *If the user or any other process do not kill it, when does 'example.js' end its execution?*

a As soon as it has shown an error on the screen.

b Approximately, two seconds after showing its first console.log message on the screen.

c As soon as it shows its last console.log message on the screen.

d It never ends since it is continuously generating periodic events.

6 *Wikipedia is a good example of a scalable service that needs many servers. How are those servers arranged in components to provide that scalability?*

a There is a single component; so, each request is processed by a single server, since this reduces to a minimum the perceived response time.

b They define several specific components that interact among them. This, combined with caching, sharding and load balancing provides scalability.

c They are structured in multiple replicated components. Clients may forward their requests to the closest component, and the latter replies without any communication with other components, minimising thus the response time.

d There is a single component, and that component has been replicated to ensure service availability. That availability also guarantees scalability.

7 *Let us consider a pair of client and server programs whose communication is implemented using the Node.js 'net' module. If multiple processes that run the client program have sent at a time their messages to the server, may the latter handle those messages without using its event queue?*

a Yes, since the event queue is not needed for handling 'net' messages.

b No, because that module uses events to notify when a connection has been set or a message has been delivered.

c Yes, since the server program may have been written using promises and in that case the event queue is not used.

d No, because the event queue is only used by promises and the 'net' module uses them.

8 *What is shown on the screen when we run this program?*

```
function f(x) {
  return function (y) {
    x = x + y
    return x
  }
}
g = f(1)
process.stdout.write(g(2)+"")
process.stdout.write(g(3)+"")
```

a 36

b 34

c 12

d 23

9 *There are two processes A and B that run in the same computer and use ØMQ sockets. The computer IP address is 158.42.60.120. A has successfully used this statement to initiate a connection with B: so.connect('tcp://158.42.60.120:8888'). Which statement should use B in order to set that A-B connection?*

a No other choice is valid.

b so.connect('tcp://158.42.60.120:8888')

c so.connect('tcp://*:8888')

d so.connectSync('ipc://158.42.60.120:8888')

10 *In a chat service we need to combine two simple unidirectional communication patterns. Let us call them A and B. A forwards the messages from each client process to the server, while B sends server messages to every client. If we need to minimise the number of send() operations to be used, which ØMQ communication pattern (or patterns) is the best for implementing A and B?*

a With a single REQ-REP we may implement A and B simultaneously.

b A: PUB-SUB, B: PUSH-PULL.

c With a single PUB-SUB we may implement A and B simultaneously.

d A: PUSH-PULL, B: PUB-SUB.

11 *What is shown on the screen when we run this program?*

```
function f(x) {
  return function (y) {
    x = x + y
    return x
  }
}
g = f(1)
h = f(1)
process.stdout.write(g(2)+"")
process.stdout.write(h(3)+"")
```

a 23

b 45

c 34

d 36

12 *Which is the choice that describes the behaviour of the following program?*

```
var i = 1
do {
  let k = i
  setTimeout(()=>{console.log(k)},
     k*1000)
  i++
} while (i<11)
console.log("End of program. i: " + i)
```

a It writes first 'End of program. i: 11' and later writes values 1 to 10, one per line, and one per second.

b It writes first 'End of program. i: 11' and later writes 11 ten times, one per line, and one per second.

c It writes values 1 to 10, one per line, and one per second. Once this has terminated, it writes immediately 'End of program. i: 11'.

d No other choice is valid.

13 *Let us consider a client and a server program that use a ØMQ REQ-REP communication pattern. The client uses this statement, so.send(['s1','s2']), to send a given request to the server. Which are the message contents actually transmitted from the REQ socket to the REP socket?*

- **a** Two segments: the strings 's1' and 's2'.
- **b** Four segments: the sender identity, the empty string, the string 's1' and the string 's2'.
- **c** Three segments: the empty string, the string 's1' and the string 's2'.
- **d** A single segment with the result of this call: JSON.stringify(['s1','s2']).

14 *Unit 3 explains that ØMQ provides weakly persistent communication (i.e., the sender keeps the messages if the receiver is not ready yet). Is there any ØMQ communication pattern unable to provide that weak communication persistency in its channels?*

- **a** No. All patterns are weakly persistent.
- **b** Yes, the REQ-REP pattern.
- **c** Yes, the PUSH-PULL pattern.
- **d** Yes, the PUB-SUB pattern

15 *Let us consider this JavaScript program:*

```
function f2 () {
  for (let let_i=0; let_i<5; let_i++) {
    console.log ("let_i: " + let_i)
  }
  console.log ("end --> let_i=" + let_i)
}

f2 ("new value")
```

What is the last line shown on the screen when we run that program?

- **a** new value
- **b** end –> let_i=5
- **c** let_i: 4
- **d** An error message (e.g., 'ReferenceError: let_i is not defined').

16 *Let us consider this JavaScript program:*

```
function f1 (a,b,c) {
  console.log ("Receiving " + arguments.length + "
arguments")
  return a+b+c;
}
let vector = [1,2,3,4]

console.log ("resultv1: " + f1 (...vector))
```

What is shown on the screen when we run that program?

- **a** Receiving 4 arguments
  resultv1: 6
- **b** Receiving 1 arguments
  resultv1: [1,2,3,4]undefinedundefined
- **c** Receiving 3 arguments
  resultv1: 6
- **d** An error message.

17 *In the second session of Lab 1, we implemented a program that generated an infinite series of stages (implemented within the 'stage' function) that took a random time (between 2 and 5 seconds per stage, with a duration generated by the 'rand' function). If only 6 stages were needed, instead of an unbounded number of them, choose the correct statement about the adequacy of the following program to solve that problem:*

```
t = 0
for (let i=0; i<N; i++) {
   t = t + rand();
   setTimeout(stage, t);
}
```

- **a** The program does not solve that problem. To fix it, it must call setInterval instead of setTimeout.
- **b** The program cannot handle the proposed task, since the second argument of setTimeout should be derived from the value of 'i'.
- **c** The program correctly solves that problem if the value of N is 6.
- **d** The program correctly solves that problem if the value of N is 5.

Partial 2

18 *This is the docker-compose.yml file to be used in the second half of the first session of Lab 3 to deploy a CBW system:*

```
version: ’2’
services:
  cli:
    image: client
    build: ./client/
    links:
     - bro
    environment:
     - BROKER_HOST=bro
     - BROKER_PORT=9998
  wor:
    image: worker
    build: ./worker/
    links:
     - bro
    environment:
     - BROKER_HOST=bro
     - BROKER_PORT=9999
  bro:
    image: broker
    build: ./broker/
    expose:
     - "9998"
     - "9999"
```

Which are the other files where the BROKER_HOST **environment variable** *is used?*

a The client.js and worker.js JavaScript programs.

b The broker.js JavaScript program.

c All choices are correct.

d The Dockerfiles for client and worker.

19 *This is a feature that usually enhances the scalability of a distributed service:*

a Strong consistency model.

b All choices are true.

c Usage of reverse proxies.

d Frequent need of interprocess coordination.

20 *Let us consider that this Dockerfile has been written for creating an image called ’broker2’:*

```
FROM tsr-zmq
COPY ./tsr.js tsr.js
RUN mkdir broker
WORKDIR broker
COPY ./broker.js mybroker.js
EXPOSE 9998 9999
ENTRYPOINT ["/usr/bin/node","mybroker","9998","9999"]
```

Can we create ’broker2’ using an appropriate command?

a Yes, since there is no error in its contents if the mentioned files exist and they are in the intended locations.

b No, because a Dockerfile cannot use ENTRYPOINT. It should use CMD, instead.

c Yes, with this command:

```
docker run broker2
```

d Yes, with this command:

```
docker commit broker2
```

21 *The first session of Lab 3 starts with* **a manual deployment** *of a broker-worker-client system. In that system, the broker is started first, and the other two components need to know* **the IP address of the broker container**. *In that manual deployment, where is that IP address used?*

a In some of the instructions of the docker-compose.yml file.

b In the CMD or ENTRYPOINT instruction of the corresponding components´ Dockerfile.

c In some arguments of the docker-compose up command.

d No other choice is correct.

Let us consider these JavaScript programs:

```
// Program: sender.js
const zmq = require( 'zeromq' )
const rq = zmq.socket( 'push' )
rq.connect( 'tcp://127.0.0.1:8888' )
rq.send(' Hello' )
```

```
// Program: receiver.js
const zmq = require( 'zeromq' )
const rp = zmq.socket( 'pull' )
rp.bindSync( 'tcp://127.0.0.1:8888' )
rp.on('message', function( msg ) {
  console.log( 'Received: ' + msg )
})
```

Note that these programs use the PUSH-PULL communication pattern. We plan to change both programs to use DEALER and ROUTER sockets instead, providing the same functionality.

22 *The changes to be applied on 'receiver.js' are:*

a

```
const rp = zmq.socket( 'dealer' ) // Line 3
```

b

```
const rp = zmq.socket( 'router' ) // Line 3
rp.on('message', function( who, msg ) { //  Line 5
```

c The proposed change does not make sense. The DEALER-ROUTER pattern cannot manage that kind of communication.

d

```
const rp = zmq.socket( 'router' ) // Line 3
```

23 *The changes to be applied on 'sender.js' are:*

a

```
const rq = zmq.socket( 'router' ) // Line 3
```

b The proposed change does not make sense. The DEALER-ROUTER pattern cannot manage that kind of communication.

c

```
const rq = zmq.socket( 'dealer' ) // Line 3
```

d

```
const rq = zmq.socket( 'dealer' ) // Line 3
rq.send( [ rq.identity, 'Hello' ] ) // Line 5
```

24 *Let us consider that this docker-compose.yml is in directory /home/user/docker:*

```
version: '2'
services:
  svca:
    image: imga
    links:

      - svcb
    environment:
      - B_HOST=svcb
  svcb:
    image: imgb
    links:
      - svcc
    environment:
      - C_HOST=svcc
    expose:
      - "9999"
  svcc:
    image: imgc
    expose:
      - "9999"
```

Please, choose the TRUE sentence about the service to be deployed using that file.

a The components in that service will start in this order: svca, svcb, svcc.

b That service can be deployed, but its components svcc and svcb will listen to same host port (9999) at the same time raising a conflict.

c We may deploy two instances of every component using this command at /home/user/docker:

```
docker-compose up -d --scale svc=2
```

d Using that docker-compose.yml, we cannot use any Dockerfile to build the missing images if any of them does not exist when we run

```
docker-compose up
```

Let us assume that image 'tsr-zmq' exists and has the contents and functionality explained in Unit 4 and Lab 3. Let us also assume that this Dockerfile (to be referred to as 'Dockerfile A', although its actual name is 'Dockerfile') has been saved in directory /home/user/docker/config:

```
FROM tsr-zmq
COPY ./tsr.js tsr.js
RUN mkdir broker
WORKDIR broker
COPY ./broker.js mybroker.js
EXPOSE 9998 9999
CMD node mybroker 9998 9999
```

25 *Let us consider Dockerfile A. Why does it use its instructions RUN and WORKDIR?*

a To ensure that the **mybroker** program runs at the root directory of the generated containers.

b To place tsr.js in the parent directory of **mybroker.js** since that is the location assumed in the code of the latter program.

c To ensure that **broker.js** (at the host) and **mybroker.js** (at the image) are placed in the same location at their respective computers.

d No other choice is correct.

26 *Let us consider Dockerfile A. Why does it use the 'EXPOSE 9998 9999' instruction?*

a To map port number 9999 in the container to port number 9998 in the host.

b To map port number 9999 in the host to port number 9998 in the container.

c To notice that the process to be run in the container will attend connections on its ports 9998 and 9999.

d No other choice is correct.

27 *Let us consider Dockerfile A. Which files are needed for building an image using that Dockerfile?*

a tsr.js and broker.js, both placed at the same location than that Dockerfile.

b tsr-zmq

c tsr.js, client.js, broker.js and worker.js

d tsr.js and mybroker.js, both placed at the same location than that Dockerfile.

28 *If we compare active and passive replication, the active model is the preferred replication model when operations modify a large part of the service state because":*

a The passive model must send those updates to the back-up replicas and the latter should apply them, while in the active model no update transfer is needed.

b When those updates are small, the active replication model exchanges those updates among all the replicas.

c When updates are large, the passive model must generate many threads of execution to apply them at the primary replica, and this blocks the execution at the primary.

d No other choice is true.

29 *Which of these alternatives is a correct difference between the multi-master and the active replication models?*

a The multi-master model may use a different processing replica (i.e., the master) per client, while the active model always uses all replicas.

b In the active model each request is only forwarded to the primary replica, while in the multi-master model each request is broadcast by the client to every replica.

c The multi-master model can handle the arbitrary failure model, while the active model cannot.

d No other choice is true.

30 *This is the docker-compose.yml file used in the last session of Lab 3:*

```
version: '2'
services:
  mariadb:
    image: docker.io/bitnami/mariadb:11.1
    volumes:
      - 'mariadb_data:/bitnami/mariadb'
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - MARIADB_USER=bn_wordpress
      - MARIADB_DATABASE=bitnami_wordpress
  wordpress:
    image: docker.io/bitnami/wordpress:6
    ports:
      - '80:8080'
      - '443:8443'
    volumes:
      - 'wordpress_data:/bitnami/wordpress'
    depends_on:
      - mariadb
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - WORDPRESS_DATABASE_HOST=mariadb
      - WORDPRESS_DATABASE_PORT_NUMBER=3306
      - WORDPRESS_DATABASE_USER=bn_wordpress
      - WORDPRESS_DATABASE_NAME=bitnami_wordpress
volumes:
  mariadb_data:
    driver: local
  wordpress_data:
    driver: local
```

Do we need to use any Dockerfile to deploy this service?

a Yes, we need to download the MariaDB and WordPress´ Dockerfile from the Bitnami site to manually build their corresponding Docker images.

b Yes, we need to download from hub.docker.com the MariaDB and WordPress´ Dockerfile to manually build their corresponding Docker images.

c No, since the 'mariadb' and 'wordpress' images are already in the local repository of our computer by default when we install Docker.

d No, the needed images are automatically downloaded from the correct remote repository when we run the **docker-compose up** command.

31 *Among other things, to run a Docker container on a given host computer, we need ...*

a A host operating system like that assumed in the image to be run.

b The **docker commit** command for starting the container.

c A previous execution of the **docker images** command for building the intended image.

d A host operating System other than Windows 11.

32 *The second session of Lab 3 proposes the deployment of another type of client (an external client) that will run in another computer, different to the host where docker and docker-compose manage the CBW containers. Which is the IP address this external client connects to?*

a That of the broker container.

b That of the worker container.

c That of the host computer where the broker container is running.

d That of the host computer where the client container is running.

33 *If we compare the causal and sequential consistency models, which of them is more relaxed than the other?*

a Their degree of relaxation are equivalent.

b Sequential

c Causal

d Their degree of relaxation cannot be compared.

34 *If we consider the CAP theorem, which of these consistency models may be respected when availability is needed in a partitioned system?*

a Strict

b No other choice is correct

c Sequential

d Any slow model

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A

DNI NIE PASSPORT

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9

ETSInf - NIST

Retake - 08/02/2024

Surnames ....

Name ....

Mark like this ▬

Like this do not mark

NOT ERASING, correct with concealer

First Partial

	a	b	c	d
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				

Second Partial

	a	b	c	d
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				