TSR: Second Partial

This exam consists of 20 multiple choice questions. In every case only one answer is correct. You should answer in a separate sheet. If correctly answered, they contribute 0,5 points to the exam grade. If incorrectly answered, the contribution is negative: -0.167. So, think carefully your answers.

THEORY

1. Service deployment requires the initial installation and configuration of a service application. Besides those tasks, deployment also includes other tasks like...

A	Program debugging. False. The debugging stage should be carefully applied and completed before any software version of an application is released. Deployment comprises all the steps to be considered after software release. Therefore, program debugging is not included in a service deployment.
B	Service lifecycle management. True. Service lifecycle management includes all tasks applied to a software application once it has been installed (e.g., activation, deactivation, removal…). Those tasks belong to the set of activities to consider in a service deployment.
C	Program development. False. As in the first sentence of this question, development must be completed before an application is released. Therefore, that task is not included in a service deployment.
D	Program design. False. Design should precede development and debugging. Then, it is not a subtask of service deployment.

2. The goal of dependency injection is...

A	To resolve inter-component dependencies using configuration files. False. Dependency injection consists in resolving inter-component dependences in a deferred and transparent way, with a minimal (ideally none) intervention of the system administrator. Configuration files demand human intervention. Therefore, they should be avoided in a dependency injection approach.

B	To eliminate all inter-component dependencies at design time. False. Components need to know different data (endpoints, interfaces,…) about the other components in order to interact with them. Some of these data are only known at deployment time (e.g., endpoints of the other components) or once the component is already running (e.g., the new endpoint of a migrated component, the database shard responsible of a given key, etc.).
ℂ	To keep dependency resolution as transparent as possible for the component developer. True. Dependency injection consists in providing the dependency resolution data at runtime, dynamically linking the component to objects that hold that information. Those objects behave as proxies: they provide the appropriate interface and give a local view for those interactions with other external components, hiding all details related to endpoint address resolution. Thus, the component code doesn't include explicit fragments for reading configuration files or accessing any other static external data.
D	To avoid the usage of containers, since they introduce performance overheads. False. Containers automate some of the dependency resolution steps. Thus, they are a help for implementing dependency injection techniques.

3. Let us assume a stateful service that needs 400 ms to locally process each update request, updating a set of data that can be transmitted to other nodes in 20 ms and applied there in 30 ms. A request message may be broadcast (in total order) in that network in 3 ms. A read-only request may be locally served in 20 ms. The ratio of accesses is 80% reads and 20% writes.
In order to scale out that service, the best approach is...

A	To replicate it following the active model. False. Using the active model, all replicas should use 400 ms in order to locally process each update request. The passive model only introduces that workload in the primary replica, while the secondary replicas should deal with 30 ms of local processing in order to apply those updates. Because of this, secondary replicas have approximately 1/13 of the workload of an active or primary one regarding update request processing. Thus, if we have a service deployed in 4 replicas, all replicas in the active model and the primary one in the passive model have a heavy workload for dealing with update requests. That workload is very low in the secondary replicas of the passive model. Those secondary replicas may be used as primary ones for other replicated services, since a large percentage of their processing capacity remains free. Therefore, the active model is not a good approach for this replication scenario.

B	To replicate it using the passive model, forwarding all requests to the primary replica. False. Although this alternative is better than the first one, the third one is better than this.
ℂ	To replicate it using the passive model, but allowing that backup replicas also reply to read-only requests. True. If the primary replica manages update requests and the secondary replicas manage read-only requests, the workload is balanced in a better way than in the two previous proposals. Recall that in the active model all requests are forwarded to every replica in order to overcome arbitrary failures. Therefore, no workload balancing is done in the active model or in the strict passive model, where all requests are processed only by the primary replica.
D	To not replicate it at all, since replication introduces too much coordination and does not allow any efficient scaling. False. Distributed services should be fault tolerant. No replication means a single point of failure and compromises service availability. Although replication introduces some level of coordination, read-only requests may be distributed among the existing replicas (see the previous sentence in this question) and, in the end, this makes scaling possible and efficient. Without replication, a single instance should deal with all kinds of requests. Therefore, that instance may become easily overloaded.

4. In data-centred consistency models, we may say that model A is stronger than B in the following cases:

A	A: causal, B: cache. False. The causal and cache consistency models cannot be compared. There are executions that are causal but not cache, and there are other executions that are cache but not causal. Model A is stronger than B when all executions in A comply with the conditions from B and there are executions that respect B but not A.
B	A: FIFO, B: cache. False. The FIFO and cache consistency models cannot be compared. There are executions that are FIFO but not cache, and there are other executions that are cache but not FIFO.
C	A: causal, B: sequential. False. Sequential is stronger than causal.
𝔻	A: causal, B: FIFO. True. Causal is stronger than FIFO. This means that all causal executions are also FIFO (since causal implies FIFO), but there are some FIFO executions that aren't causal. For instance, in a system with 3 processes, this execution… W1(x)1, R2(x)1, W2(x)2, W1(x)3, R2(x)3, R3(x)2, R3(x)1, R3(x)3, R1(x)2, … …is FIFO since P1 has written two different values (1 and 3, in that order) onto "x" and P3 and P2 have received both values in that writing order. Since no other process has written more than one value, this means that all conditions for FIFO consistency have been respected in that execution. However, P2 wrote value 2 once it had read value 1. This means that both values are causally related (1 → 2) but P3 hasn't obtained them in that causal order. Therefore, this execution isn't causal and this proves that FIFO is more relaxed than causal (and, equivalently, that causal is stronger than FIFO).

5. Scalable NoSQL datastores do not support the relational model because...

A	The relational model does not admit replication. False. Databases supporting the relational model may be replicated, as any other software service may be. For instance, there are several releases of MySQL, Oracle or Microsoft SQL Server (among other providers) that replicate their databases.
B	The relational model cannot support sharding. False. Relational databases may be sharded. Indeed the first sharding proposals were made assuming replicated relational databases.
C	Relational data must be kept on disks. False. Of course, data should be kept on disks, but that is not a constraint for using replication or being scalable.
𝔻	Relational transactions may demand complex concurrency control mechanisms. True. The transactions being used in a relational database become complex when replication is introduced. In order to commit a transaction a distributed algorithm is needed (2PC or 3PC) and that algorithm demands a lot of interaction among the transaction participants. Additionally, concurrency control mechanisms may demand their extension to a distributed management (distributed locks, for instance) and, again, this demands a lot of interaction among the participating processes. Because of this, NoSQL datastores have renounced to this kind of transactions in order to be scalable.

6. The CAP theorem...

A	...requires that scalable and available services use always a strict consistency model in order to tolerate network partitions. False. On the contrary, the CAP theorem states that strict consistency cannot be maintained in services that should be highly available while network partitions arise.

𝔹	...allows scalable services to relax their consistency while the network remains partitioned, ensuring in this way their availability. True. The CAP theorem says that distributed services cannot simultaneously support strong consistency, availability and network partition tolerance. Thus, when the network becomes partitioned, either availability or strong consistency should be sacrificed. If a service prefers availability, it must relax its replica consistency while the network remains partitioned.
C	...doesn't tolerate the implementation of highly available services using strong consistency models. False. Strong consistency models may be used in distributed services. What is being said in the CAP theorem is that if a service wants to remain strongly consistent while the network is partitioned, then it should renounce to its complete availability. This means that minor subgroups cannot reply to their clients and they remain unavailable. Indeed, CAP also tolerates complete availability and strong consistency. In that case, the service must be deployed in a way that guarantees that no network partition may arise among its replicas. For instance, in small deployments for a reduced set of clients with a few replicas in a single lab with physical network replication.
D	...doesn't make sense in cloud computing data-centres, since no network partition may happen in those systems. False. Cloud computing tries to provide the image of unlimited scalability. To this end, multiple data-centres may be used in order to deploy a service. Even in case of using a single data-centre, that data-centre consists of many racks of computers and many network segments. In those cases, network partitions may arise. Therefore, the constraints stated in the CAP theorem apply in those cases.

7. Regarding service scalability...

A	A service cannot be scaled both horizontally and vertically. False. Horizontal and vertical scalability aren't mutually exclusive.
B	Decentralised algorithms improve scalability on distance. False. Decentralised algorithms improve scalability on size, but not necessarily scalability on distance.
C	Sharding improves administrative scalability. False. Sharding improves scalability on size, but doesn't simplify administrative scalability.
𝔻	Contention avoidance is a key factor to achieve service scalability. True. Contention prevents services from scaling since it blocks service execution. Therefore, contention avoidance is one of the main principles for enhancing service scalability.

8. The main goals of a security subsystem are:

A	Protection, access control and physical security. False. Access control and physical security aren't goals. They are security mechanisms that may be used for supporting concrete security policies.
B	Protection, trust management and cryptographic support. False. Cryptography is a mechanism that may be used for enhancing the confidentiality and integrity of a system. Trust management is a means (i.e., mechanism) for ensuring and evaluating the correctness of a security system. They aren't security goals, but security mechanisms.
C	Accountability, integrity, confidentiality and availability. True. These are the four goals that we have presented in Unit 8 for security subsystems.
D	Robust policies, efficient mechanisms and correct assurance. False. These are the three types of security tools being needed for specifying, implementing and evaluating the correctness of a security system, respectively.

SEMINARS

9. Which of the following tasks is <u>NOT</u> done when we use this docker command?

```
docker run -it ubuntu /bin/bash
```

A	Run the /bin/bash command in a container. False. The last argument in that command line states that the program to be run in the generated container must be "/bin/bash".
B	Download the "ubuntu:latest" image from the Docker Hub if it wasn't in the local docker repository. False. The "ubuntu" argument is the name of the image to be used for starting this container. When no tag is given, Docker assumes the "latest" tag. Therefore, the image being looked for is "ubuntu:latest". Docker searches that image in the local repository. If it isn't there, Docker downloads it from the Docker Hub and keeps it in the local repository once downloaded.
C	Collect the output of the container being used for executing that command. That output can be shown using the `docker logs` command. False. The container output is collected by the docker daemon and it can be shown using the "docker logs" command using the container ID or name as its argument.
D	Remove automatically this container once its execution has terminated. True. To behave in that way, the command line should include the "--rm=true" option. Otherwise, the container is kept once it completes its execution.

10. The `docker commit a b` command…

A	Creates a new container called "a" using the Dockerfile placed in folder "b". False. New containers are generated with either the "docker run" or "docker create" command.

B	Creates a new image "b" using the Dockerfile placed in folder "a". False. In order to generate a new image from a given Dockerfile, we should use the "docker build" command.
ℂ	Creates a new image "b" with the current contents of the container whose name or ID is "a". True. That is a valid short description for the "docker commit" command.
D	Commits a docker transaction "a" that was started with a `docker pull` or `docker push` command, generating a container ID "b". False. Docker doesn't use transactions.

11. The "cluster" NodeJS module is used for…

A	…facilitating the deployment of a set of NodeJS programs onto a cluster of computers. False. NodeJS modules are used in the development stage. They cannot be used at deployment time.
B	…managing multiple threads in a NodeJS process. False. JavaScript programs are single-threaded. It is impossible to manage multiple threads in that language and NodeJS is a JavaScript interpreter.
ℂ	…managing a set of NodeJS processes able to share some resources; e.g., a listening port and a given program to be run by all of them. True. That is the goal of that module.
D	…easily implementing multiple memory consistency models. False. We haven't used the "cluster" module to that end.

12. MongoDB uses the following mechanisms to improve its scalability:

A	Distributed concurrency control. False. Distributed concurrency control is more complex and introduces more contention than local concurrency control. Because of this, it doesn't improve scalability.
B	Decentralised algorithms. False. Although decentralised algorithms are a good mechanism for improving size scalability, we haven't described any decentralised algorithm in our descriptions of MongoDB.
ℂ	Passive replication and data sharding. True. MongoDB uses primary-backup replication (i.e., passive replication) and data sharding (increasing thus its capacity for concurrently serving multiple requests in multiple MongoDB instances) in order to be scalable.
D	Administrative scalability. False. We haven't seen any MongoDB configuration mechanism for improving its administrative scalability and, in this way, improve also its overall scalability.

13. The main goal of MongoDB configuration servers is…:

A	To behave as arbiters when a replica fails. False. The arbiter role may be played by a "mongod" server, but not by a "configuration server".

B	To take care of data distribution in multiple shards. True. This is the main goal of MongoDB configuration servers.
C	To respect the CAP theorem when network partitions arise. False. No configuration server intervention is demanded in order to manage network partitions in MongoDB. That management is done at the "replica set" scope. In those cases, only the largest subgroup is allowed to continue and only when it maintains more than a half of the replica set being considered.
D	To detect node failures, starting a recovery protocol when any failure happens. False. The MongoDB documentation doesn't describe in detail its failure detection mechanism. It is unclear which elements participate in that detection. However, that management is done at the replica set scope; i.e., internal to each replica set. This means that such detection is embedded in the interactions among "mongod" processes that belong to the same replica set.

14. Regarding the thematic classification of vulnerabilities shown in Seminar 8, it is true that…

A	The exploitation of faults in security policies cannot be as easily automated as the exploitation of weak passwords. True. Faults in security policies should be thoroughly analysed and checked by potential attackers in order to be identified and found. This demands a lot of time and interaction with that system. On the other hand, there are many lists of widely used "weak passwords". Once a user ID is known by a potential attacker, the latter may attempt a remote access using any of those lists of "weak passwords". Therefore, such kind of vulnerability exploitation can be easily automated.
B	Phishing is a "software error" vulnerability. False. Phishing is a "social engineering" vulnerability.
C	Physical protection is an example of "social engineering" vulnerability. False. Physical protection is a security mechanism. It isn't a vulnerability.
D	A fault in a personal-protection security policy demands less interaction to be exploited than a software-error vulnerability in the operating system. False. Operating system vulnerabilities are documented in many systems (for instance, when a patch is published as an upgrade to fix that vulnerability). Therefore, no previous interaction with a system with those vulnerabilities is needed in order to start an attack that exploits them. On the other hand, personal-protection security policies shouldn't be known by external agents and their faults, if any, will generally demand more interaction in order to be identified and exploited in attacks.

15. Given this Dockerfile…

```
FROM zmq
RUN mkdir /zmq
COPY ./broker.js /zmq/broker.js
WORKDIR /zmq
EXPOSE 8000 8001
CMD node broker.js
```

Which of the following sentences is FALSE?

A	We need to have a "broker.js" file in the host directory where this Dockerfile is placed. Yes. The third line in this Dockerfile assumes that such file is placed in the same directory than the Dockerfile. The relative pathnames for the first argument of a COPY command assume that directory as their origin.

B	The program to be run in the containers started from this image uses port 8000 of the container, and it is mapped to port 8001 in the host computer. False. Those port numbers appear in the fifth line of the Dockerfile. The EXPOSE command is used in that line. EXPOSE simply lists the public port numbers where the process running in this container will listen for connections. They are container ports. It doesn't set any mapping between container ports and host ports.
C	By default, the containers generated from this Dockerfile will run the **node broker.js** command. Yes. This is stated in the last line of the Dockerfile, using the CMD command.
D	This Dockerfile assumes that a "zmq" docker image exists and that image holds a correct installation of the "node" JavaScript interpreter. Yes. The first Dockerfile line states that the image to be used as a base is called "zmq". The last line uses the "node" interpreter. Therefore, we may assume that "zmq" includes a correct installation of that interpreter, since none of the remaining lines tries to install "node".

16. Let us assume that the broker component shown in question 15 is now included in a docker-compose.yml file with these contents (among others that refer to other components):

```
version: '2'
services:
...
 bro:
  image: broker
  build: ../broker/
```

Which of the following sentences is FALSE?

A	We may start at least one instance of the broker component with the command **docker-compose up –d** True. The "docker-compose up" command starts one instance (or more, if they have been previously stopped) of each of the components mentioned in the **docker-compose.yml** file.

B	Once the service is started, we may run 5 instances of the broker component using **docker-compose scale broker=5** False. That component is called "**bro**" in that file. Therefore, the appropriate command line to be used is **docker-compose scale bro=5**
C	This "docker-compose.yml" file assumes that the Dockerfile shown in question 15 is placed in the folder "../broker/". True. That is the goal of its "**build**" command: to refer to the location of the Dockerfile being needed for building that image.
D	We may use **docker-compose stop bro** in order to stop all instances of this broker component. True. That is the docker-compose command being needed to this end.

17. Let us assume a replication protocol based on a sequencer process that uses a PUB ZeroMQ socket to forward all write events to the participating processes, in order of reception. Those events are communicated to it using PUSH-PULL channels connected to a single PULL socket held by that sequencer. That replication protocol supports the following consistency models…

A	Only the strict one. False. The strict consistency model demands a high degree of coordination in order to be implemented. ZeroMQ sockets implement asynchronous communication. Therefore, a simple implementation based on ZeroMQ sockets won't ensure strict consistency.
B	Only the cache one. False. The implementation described in this question is the one shown in Seminar 5 for implementing the sequential consistency model. Sequential consistency is stronger than cache. This means that the resulting processes also comply with the cache consistency model, but not only with cache.
C	Only the causal one. False. The implementation described in this question is the one shown in Seminar 5 for implementing the sequential consistency model. Sequential consistency is stronger than causal. This means that the resulting processes also comply with the causal consistency model, but not only with it.
D	Sequential, processor, causal, cache and FIFO. True. The implementation described in this question is the one shown in Seminar 5 for implementing the sequential consistency model. Sequential consistency is stronger than processor, causal, cache and FIFO. Therefore, the resulting processes support all these consistency models.

18. Given the following execution:
W1(x)1, R4(x)1, W2(y)2, W1(y)3, W2(x)4, R3(y)2, W3(x)5, R1(x)5, R2(x)1, R3(x)1, R4(y)3, R1(y)2, R3(y)3, R4(x)5, R3(x)4, R2(y)3, R4(y)2, R1(x)4, R2(x)5, R4(x)4.
That execution respects the following consistency models…

A	Only the FIFO one. True. There are five write actions in this execution. Two of them have been done by P1 (value 1 to "x", value 3 to "y", in that order), two other by P2 (value 2 onto "y" and value 4 on "x", in that order) and another by P3 (value 5 on "x"). FIFO consistency

	demands that values written by a process are read in that writing order by the remaining processes. This means that value 1 should precede value 3, and value 2 should precede value 4. This has happened in all processes. Those two pairs of written values may be interleaved in any way in every reader. Besides this, value 5 may also be freely interleaved, since it has been written by another different process. In our execution, each process has seen the following sequences of values: P1: 1, 3, 5, 2, 4 P2: 2, 4, 1, 3, 5 P3: 2, 5, 1, 3, 4 P4: 1, 3, 5, 2, 4 Therefore, all processes respect the FIFO consistency model. Let us see whether they comply with other consistency models. To this end, we should check first the cache and causal models. Cache is not comparable to FIFO and more relaxed than processor (that is more relaxed than sequential). Causal is stricter than FIFO but more relaxed than sequential. Therefore, if cache is not supported, processor cannot be; and if causal is not supported, sequential cannot be. Cache consistency demands that all processes agree on the sequence of values on each of the variables (each variable being considered separately from the rest). There are three "x" values (1, 4 and 5) and two "y" values (2 and 3). Unfortunately, there is no agreement on the values written to "x" (P1 and P4 have seen 1, 5, 4; P2 has seen 4, 1, 5; and P3 has seen 5, 1, 4). That is enough to say that the execution doesn't respect cache consistency. As a result, it doesn't respect processor or sequential, either. [Note that there is no agreement on the values written to "y", either (P1 and P4 have seen 3, 2; while P2 and P3 have seen 2, 3)]. Regarding causal consistency, there is a path of causal dependency between values 2 and 5 [W2(y)2,…, R3(y)2, W3(x)5,…] since the writer of value 5, P3, read value 2 before writing value 5. This means that all processes should see value 2 before value 5 in order to comply with causal consistency. This doesn't happen in processes P1 and P4. Therefore, this execution isn't causal. If it isn't causal, it cannot be sequential, either. Because of this, the execution only respects the FIFO model.
B	Only the cache one. False. See the explanation of the first part in this question.
C	Processor, FIFO and cache. False. See the explanation of the first part in this question.
D	Sequential, processor, causal, cache and FIFO. False. See the explanation of the first part in this question.

19. **Given the following file-downloading server program…**

```
var cluster = require('cluster');
var fs = require('fs');
var path = require('path');
var zmq = require('zmq');
var os = require('os');
const ipcName = 'Act2.ipc';
const dlName = 'ipc://'+ipcName;
if (cluster.isMaster) {
   var numCPUs = os.cpus().length;
   var rt = zmq.socket('router');
   var dl = zmq.socket('dealer');
   rt.bindSync('tcp://127.0.0.1:8000');
   dl.bindSync(dlName);
   rt.on('message', function() {
         msg = Array.apply(null, arguments);
         dl.send(msg);   });
   dl.on('message', function() {
         msg = Array.apply(null, arguments);
         rt.send(msg);   });
} else {
  var rep = zmq.socket('rep');
  rep.connect(dlName);
  rep.on('message', function(data) {
    var request = JSON.parse(data);
    fs.readFile(request.path, function(err,
data) {
     if (err) data = ' NOT FOUND';
      rep.send(JSON.stringify({
          pid: process.pid ,
          path: request.path ,
          data: data ,
          timestamp: new Date().toString()
      }))
   })
  })
}
```

We have tried to run that program, but it didn't do anything useful. Its main problem is that …

A	ZeroMQ sockets do not admit an "**ipc://**" transport. False. The "**ipc:**" transport can be used in order to communicate two or more processes placed in the same computer. It is admitted.
B	No cluster worker process has been created. True. We need to create multiple worker processes (using the **cluster.fork** function to this end) in the body of the master's code block. No instruction of this kind is in the shown version of this program.
C	We are trying to forward "cluster" internal messages through a ZeroMQ DEALER socket. False. The messages being exchanged between worker and master processes in this program user a ZeroMQ DEALER-REP channel to this end. They cannot be qualified as "internal" cluster messages. A DEALER-REP channel works correctly in ZeroMQ and it is the easiest implementation in case of using a ROUTER-DEALER pattern in the broker.
D	A server cannot use a ROUTER socket as its endpoint. False. We have seen many examples of replicated services that use a broker process in order to balance their incoming workload among their worker replicas. That broker process uses, in the common case, a ROUTER socket as its public endpoint.

20. **The OpenSSL Heartbleed vulnerability described in Seminar 8 is an example of vulnerability that belongs to these classes:**

A	"Fault in security policy" regarding its category and "human staff" regarding its origin. False. See part B.

B	"Software error" regarding its category and "library/middleware" regarding its origin. True. This vulnerability was caused by a software design error (the heartbeat messages in the SSL protocol required an answer of a sender-specified size, and that size had a very large bound, thus obtaining a snapshot of the replying process memory that could hold important information; e.g., passwords) in a library (OpenSSL). Therefore, its category is "software error" and its origin is "library/middleware" since that problem affected the programs using those vulnerable versions of the OpenSSL library.
C	"Social engineering" regarding its category and "developer" regarding its origin. False. See part B.
D	"Software error" regarding its category and "human staff" regarding its origin. False. See part B.