TSR / NIST

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. In the scope of deployment configuration, this is an example of "dependency injection":

a	To throw an exception when the port to be used is already assigned to another (i.e., to a different and already running) server. False. That fact is not related with component dependences. A component A depends on another component B if A uses the operations provided by B. In order to use those operations the service endpoint of B should be known. The term "dependence" refers to that fact: a client component A needs to know and use the endpoint where a service B waits for incoming requests.
b	To start another server replica when the workload is increasing; i.e., to manage a scale-out action. False. Again, that fact is not related with dependences.
c	To dynamically learn (or even update) which is the endpoint of another service component that will be used later on. True. In this subject we have used the term "dependence injection" in order to refer to that scenario.
d	Some (optional) component of the service is not deployed (i.e., it has no deployed instance). False. If that part of the functionality of the service has not been installed in a deployment, that fact may be adequately configured elsewhere. In that case, the remaining service components do not need to call its services. Therefore, those other components cannot behave as its clients and they will not depend on it. Thus, no dependency injection is needed.

2. Which of the following tasks IS NOT a deployment-related task:

a	Component installation in the target hosts. Software installation is included in service deployment.
b	Application debugging. Deployment refers to the set of tasks to be driven once an application has been released and is ready to be installed and used. Debugging must be done before publishing an application.
c	Dependency resolution. This task is included in service deployment. It is one of the last tasks to be completed in the installation and configuration procedures in order to intercommunicate different service components.
d	Software upgrading. This has also been considered a deployment-related task. When software is upgraded, it usually needs to be reinstalled (at least, partially) and, in some cases, reconfigured.

3. Let us imagine that we have written an algorithm AL1 in order to reach consensus in a distributed system assuming the Byzantine failure model. Then:

a	AL1 is useless, since no real system behaves in that way (i.e., in a way that matches

	the Byzantine failure model). False. Although, at a glance, this failure model may seem too much pessimistic since usually the pieces of software to be used in a real system behave correctly, its assumptions are not unrealistic. In many cases, software may have bugs and, from time to time, its behaviour does not comply with its specifications. In those scenarios, an algorithm designed assuming an arbitrary failure model may be able to behave correctly even when some of its pieces do not run as initially expected.
b	AL1 needs to replicate all its processes using the passive model, since the Byzantine failure model can only be managed using that replication model. False. Processes replicated with the passive model cannot manage arbitrary failures.
ℂ	AL1 is easier to port to a real system than another algorithm AL2 that assumes the stop failure model, since the assumptions of AL2 are hard to reach in real systems. True. The stop failure model assumes a perfect behaviour of the processes until they fail by stopping and remaining stopped. This means that their software must be perfect, without any error, and that the failures, when they arise, may only stop that correct process, without generating any kind of incorrect message or output while failing. This is extremely difficult to guarantee in a real system. On the other hand, the arbitrary (or Byzantine) failure model assumes the worst possible behaviour (unexpected states and behaviours from participating processes). This means that, in the common case, the real system will behave in a better way than initially assumed.
d	AL1 will be one of the simplest algorithms, since the Byzantine failure model simplifies failure handling in the algorithms that assume it. False. On the contrary, the Byzantine failure model is the most challenging one at algorithm design time.

4. The active replication model...

ⓐ	…may handle replica failures faster than the passive replication model. True. In the active replication model all replicas have the same role. If one of them fails, the remaining ones are able to serve the requests initially targeted to the failed replica without needing any reconfiguration. As a result of this, failures are handled in the fastest and easiest way.
b	…is unable to handle the Byzantine failure model. False. It may handle Byzantine failures, since every active replica answers to the client and the client may compare all those answers before accepting a reply to its request. The passive replication model does not use any answer redundancy. Because of this, it cannot handle arbitrary failures.
c	…cannot ensure sequential consistency. False. In the common case, its replicas are sequentially consistent. All requests are delivered in the same order to every replica. If each replica handles those requests in a FIFO order, then the resulting consistency is sequential.
d	…is used by default in the MongoDB replica sets. False. MongoDB replica sets can only use the primary-backup (or passive) replication model. There is no way of changing its replication model.

5. Considering the constraints of the CAP theorem, if we manage network partitions using the primary partition model then...

a	…sequential consistency may be ensured in minor process subgroups. False. The primary partition model compels every minor subgroup to stop. Therefore, they remain unavailable and their consistency is not relevant. Thus, they are not consistent with the state managed at the subgroup with a majority of processes.
b	…sequential consistency may be ensured in the process subgroup with a majority of processes. True. All replicas in that "big" subgroup are strongly (i.e., sequentially) consistent among them. The remaining replicas remain stopped. This means that they are considered "faulty" and no request may be processed by them.
c	…availability is ensured, and replica consistency is relaxed. False. Availability is lost in all minor subgroups. Replicas in those subgroups are not considered. Consistency may be still strong among the live replicas (i.e., those in the biggest group if it exceeds a half of the system processes).
d	…availability is lost in the major process subgroup. False. That subgroup is the unique one that remains available.

6. Considering the constraints of the CAP theorem, if we must ensure availability in a service that is deployed in multiple datacentres where network partitioning should be tolerated, then…

a	…we may use sequential consistency among all service replicas. False. Sequential consistency demands that every process sees the same sequence of values (considering all variables in a joint way). If the network remains partitioned and every group of processes remains available, each of those groups may generate a different sequence of values. Therefore, sequential consistency cannot be guaranteed in that scenario.

b	…we should use a relaxed consistency model; e.g., FIFO consistency. True. FIFO consistency only demands that the writes generated by each process are seen by the remaining processes in the order they have been written. That is a per-writer order with freedom for mixing the values generated by different processes. While the network remains partitioned, that ordering may still be respected, since there is no requirement on timeliness (i.e., on obtaining the most recent value in each read).
c	...there is no constraint on replica consistency. False. Strong consistency models (e.g., those requiring consensus on a single sequence of values, either globally or per variable) cannot be maintained while the network remains partitioned if every live process remains available.
d	...service availability cannot be ensured in a scenario like that, consisting of multiple datacentres geographically dispersed. False. All services and processes may remain available if they assume a relaxed consistency model (e.g., FIFO or causal) that tolerates a delayed propagation of writes. In those models, write propagation is resumed once network connectivity is restored.

7. Which of these services is the most elastic?

a	One that adds a replica per hour, independently on the workload at that moment. False. In order to be elastic, a service must be scalable and adaptive to the current workload, provisioning an adequate amount of resources (i.e., minimising economical costs). Besides, its adaptability must be managed in an autonomous way. In this example, the service being described seems to be scalable, but it is not adaptive. Therefore, it is not elastic.
b	One that reports its currently supported workload, remaining capacity and amount of replicas to the system administrator, who decides the scaling action to be applied. False. In this case, its management is not autonomous. Therefore, it is not elastic, either.
c	One that autonomously adapts its number of replicas to the current workload, maintaining an appropriate quality of service (as stated in its SLA). True. This description matches the requirements for elastic services.
d	One that never monitors its resource usage. False. Resource monitoring is needed for implementing adaptability techniques that do not overprovision resources. Thus, this service cannot be elastic.

8. Which of these approaches DOESN'T increase the scalability of a service?

a	Task distribution. False. Task distribution is one of the best mechanisms for improving service scalability. The workload to be supported is balanced among a potentially large set of processes. If the distribution is correctly made, no concurrency control mechanisms are needed and the resulting performance is directly proportional to the amount of processes.

b	Data distribution. False. Again, data distribution improves scalability, since it avoids concurrency problems when multiple clients access disjoint data parts.
ℂ	To take decisions using voting techniques among all participating processes. True. Voting techniques demand interaction and synchronisation among processes, delaying the start of subsequent processing stages in the algorithm they take part of. This is a strong penalty for throughput and scalability.
d	To use caches. False. If the results of previous operations are cached by clients or reverse proxies, subsequent requests for the same operations may get their results from that cache. With this, server workload decreases and scalability increases.

TSR / NIST

SEMINARS

9. Which of these characteristics distinguishes virtual machines from Docker containers?

a	Some applications to be run on them may require libraries that are not present in the host operating system. False. This is supported in both kinds of tools.
b	The need of a host operating system. False. In the common case, both need a host operating system.
c	The isolation from other containers and/or virtual machines. False. Both provide a good degree of isolation from other instances.
d	Virtual machines can run applications that cannot be run in the host operating system. True. Using virtual machines the guest operating system kernel to be run in a virtual machine may be different to the host operating system. With this, we may run applications that cannot be run in any way in that host operating system. For instance, using a Linux host operating system, we may run a Windows guest system in a virtual machine, but we are compelled to use a Linux guest system in a Docker container. Every application to be run in the guest system of a container may also be run in the host operating system, installing the appropriate libraries if needed.

10. With the "docker commit" command we can...

a	End a pull- or push-repository transaction. In those transactions, a Docker image is either downloaded from or uploaded to a repository. False. There are no pull- or push-related transactions in Docker. Those image transferring actions are done in an individual way, instead of inside the context of a "transaction" that encompasses any other actions.
b	Create a Docker image, taking a Dockerfile as its base. False. The command to be used in that case is "docker build".
c	Start a Docker container, taking a Docker image as its base. False. The command to be used in that case is either "docker run" or "docker start".
d	Create a Docker image, taking a Docker container as its base. True. That is the intent of that command.

11. Let us assume that this Dockerfile is correct (i.e., no error arises when it is processed):

```
FROM zmq
COPY ./myProgram.js /server.js
EXPOSE 8000 8001
CMD node /server.js
```

Which of the following sentences is FALSE?

a	There is a "zmq" Docker image, either in the local or the global repository. True. That is the requirement of the "FROM" statement in the first line of this Dockerfile. If no error has been raised, then such requirement has been met. Therefore, the statement is true.

b	The resulting components will run a "server" process bound to port 8001 in its container. Such a port corresponds to port 8000 in its host. No. The EXPOSE line means that such “server” process is using both ports (8000 and 8001) in its container and those ports may be used by other containers being run in that same host. No correspondence is set between any of those ports and the public ports of the host, to be used from other external computers.
c	The image to be generated from this Dockerfile may run other programs besides "node server.js". True. By default, the program to be run is “node server.js” (as stated in the last line of the Dockerfile), but we may use additional arguments for the “docker run” command in order to specify that other programs should be run in those invocations. For instance, assuming that we have generated a new image called “my-image” using that Dockerfile, then this command: docker run –i –t my-image /bin/bash …will start a container from that image and will run the /bin/bash shell in it, reading its input from the terminal and showing its output in that same terminal.
d	When an image is generated from this Dockerfile, a file called "myProgram.js" should be in the folder where such Dockerfile is placed. True. That is the requirement being stated in the second line of the Dockerfile (COPY…).

12. Which of the following sentences is FALSE about the docker-compose command?

a	It assumes that a docker-compose.yml file exists in the folder where that command is run. True. That is the default assumption for that command. If we are interested in running the command in another folder, we must specify with an option (either “-f” or “--file”) the location of that file or an alternate name for it.
b	It always needs that a Dockerfile exists for each one of the components to be run. No. We may use images that have not been developed by us in order to run our services. In those cases, we cannot use the “build:” directive in the docker-compose.yml file, identifying the location of the Dockerfile to be used. Instead, we may only use the “image:” directive in order to state the name of the (already built) image to be used for deploying a given component.
c	It may be used for controlling a service based on multiple components. All instances for all those components are run in the same host. True. That is the basic goal of this command.
d	Its "scale" action (i.e., when we use the "docker-compose scale" command) may be used for changing the amount of instances of a given component in the service. True. That is the goal of that action.

13. Which of these sentences is FALSE about "fast" consistency models?

a	When a read or write action is run by a process, that action may return control to its invoker without exchanging any message with other processes. True. That is a correct and short definition for “fast” consistency models.

b	FIFO is a fast consistency model. True. FIFO is fast, since it only imposes an order on the writes generated by each writer. Thus, those writers may choose when to propagate their updates and do such propagation in a lazy way, once those write operations have returned control to its process. Its single requirement is to respect writing order in inter-process propagations.
c	Causal is a fast consistency model. True. Causal is also fast, since causal order may be implemented using FIFO lazy write propagations. To this end, those messages should carry vector clocks associated to each propagated value.
d	Sequential is a fast consistency model. False. Sequential consistency requires consensus on the order of writes before considering that a local write operation has been completed. This requires communication among all sharing replicas and breaks the definition of fast models.

14. In order to implement the FIFO consistency model (using ZMQ on TCP), we need:

a	A sequencer process that implements a total order. False. No sequencer is needed in the FIFO model, since sequencers are needed for centralising decisions in models that require consensus on value order, but the FIFO model has no requirement of that kind, as it has been explained in the previous question.
b	That each process multicasts its writes using a PUB socket and receives the writes from all other processes using a single SUB socket. True. This is an easy way to ensure FIFO order on the writes propagated by each process, as the FIFO model requires.
c	That each process multicasts its writes using a PUB socket per each variable it is able to write and a single SUB socket for receiving the writes from all other processes. No. That implementation only ensures that readers see that each process writes its values in program order on each particular variable. However, each reader may get those values on different variables with different interleavings. That behaviour is more relaxed than the FIFO model. Indeed, that model is known as “slow memory” and it has not been explained in this subject.
d	To use vector clocks and add the current vector clock to each written value when it is multicast to the other processes. No. That is not needed in the FIFO model. Their usage may be needed in the causal model, but not in the FIFO model. FIFO may be implemented using a simpler mechanism.

15. The "cluster" module in NodeJS…

a	…makes possible that a set of containers is deployed in more than one host computer. No. That module cannot handle containers nor deploy software in multiple computers.

b	…extends docker-compose commands for deploying NodeJS programs faster than without such "cluster" module. No. The "cluster" module has no relationship with the "docker-compose" command.
c	…makes possible that NodeJS programs be multi-threaded. No. NodeJS is a JavaScript developing framework. JavaScript programs cannot be multi-threaded.
d	…makes possible that a set of worker NodeJS processes share the ports that have been bound by the master NodeJS process. Yes. That is the goal of the "cluster" module.

16. We are writing a NodeJS program with the 'cluster' module. It uses as many workers as processors and the master process reports every second to the user how many requests have been served up to now by all workers.

```
var cluster = require('cluster');
var http = require('http');
if (cluster.isMaster) {
  var numReqs = 0;
  setInterval(function(){ console.log("numReqs =", numReqs);}, 1000);
  function messageHandler(msg) {
    numReqs++;
  }
  var numCPUs = require('os').cpus().length;
  for (var i=0; i < numCPUs; i++) cluster.fork();
  /* (1) Worker message management instructions should be here. */
  /* (2) Worker regeneration code should be here, if any.       */
} else {
  http.Server(function(req, res) {
    res.writeHead(200); res.end('hello world\n');
    process.send({ cmd: 'notify' });
  }).listen(8000);
}
```

Choose which instructions are needed by the master process to manage (1) the worker messages:

a	`cluster.on('message', messageHandler);` False. The cluster object is not an event manager. Thus, it cannot handle in a direct way any events using the "on()" method.

b	`for (var i in cluster.workers)`
	`  cluster.workers[i].on('message', messageHandler);`
	True. Message handlers should be managed by "worker" objects. Those entities are stored in the "cluster.workers" array kept by the master process. The event to be managed is the "message" event.

c	`for (var i in workers) workers[i].on('notify', messageHandler);` False. Message handlers should be managed by "worker" objects. Those entities are stored in the "cluster.workers" array kept by the master process. The event to be managed is the "message" event, instead of the "notify" event.
d	`cluster.on('notify', messageHandler);` False. The cluster object is not an event manager. Thus, it cannot handle in a direct way any events using the "on()" method.

17. In the program shown in question 16, we want to keep the amount of workers constant. To this end, if any of them dies, it should be replaced by a new worker. Which are the instructions (2) that implement such a functionality?

a	`cluster.on('exit', function(worker, code, signal) {` `  cluster.fork();`

	`});` The cluster.fork() operation is the one being needed for generating a new worker process. Therefore, it should be the one to be used inside this event handler.

b	`cluster.on('exit', function(worker, code, signal) {`
	`  worker.restart();`
	`});`
	The cluster.fork() operation is the one being needed for generating a new worker process. Therefore, it should be the one to be used inside this event handler.

c	`cluster.on('exit', function(worker, code, signal) {`
	`  worker.process.fork();`
	`});`
	The cluster.fork() operation is the one being needed for generating a new worker process. Therefore, it should be the one to be used inside this event handler.

d	`cluster.on('exit', function(worker, code, signal) {`
	`  worker.process.start();`
	`});`
	The cluster.fork() operation is the one being needed for generating a new worker process. Therefore, it should be the one to be used inside this event handler.

18. MongoDB uses the following scalability mechanisms:

a	Active (or state-machine) replication and task distribution. False. It does not use state-machine (i.e., active) replication. It uses primary-backup (i.e., passive) replicated, instead.
b	Primary-backup replication and data distribution (horizontal partitioning). True. It uses both mechanisms (among others) for improving its scalability.
c	Active replication and data distribution (horizontal partitioning). False. It does not use state-machine (i.e., active) replication. It uses primary-backup (i.e., passive) replicated, instead.
d	MapReduce. False. MongoDB is not directly intended for MapReduce processing.

19. Which statement is FALSE about "mongos" processes:

a	They are request forwarders. True. They behave as request forwarders, using their configuration metadata knowledge in order to balance in a proper way the workload among all existing “mongod” servers.
b	They hold a cache of the MongoDB sharding configuration metadata. True. The ask the MongoDB configuration servers when needed about such metadata. Once their replies are obtained, they are kept in a local cache until it becomes invalid due to a shard rebalancing.
c	They are placed in client computers. True. These processes are kept at the client side.
d	They may be replaced by a "replica set". False. A “replica set” may be used for replacing a “mongod” server when needed (e.g., for either enhancing its availability or its capacity of service), but not for replacing a “mongos” process.

20. Which statement is FALSE about the MongoDB "write concern":

a	It states how many replicas should persist the modifications caused by a write, delete or update operation.

	True. That is the main function of the “write concern”.
b	Lower values in the "write concern" reduce the write service time, but endanger data persistency in case of failure. True. As a result of what has been stated in the first sentence, if only a few replicas persist the data (i.e., when a low value has been set for the “write concern”) then the write service time will be minimal, but data persistency will be lost when those few replicas fail.
c	Its recommended value is "majority", since it provides a good compromise between performance and failure tolerance. True. That is the recommended value according to the MongoDB documentation. More than a half of the exiting replicas should persist the written data. This avoids problems and pauses when a few (i.e., a minority) replicas become disconnected or fail.
d	It needs a confirmation from the client for every demanded operation. False. No additional interactions with clients are needed when the value for the “write concern” is modified in a write-related operation.