**** **[Erasmus+](http://www.erasmusplus.org.ua/)**

**National Technical University of Ukraine “Igor Sikorsky KPI“**

# Course “IS Development and Deployment”

# Practice tasks for topic 11 - 14

# Kyiv-2018

**Topic 11. Server Infrastructure**

**Practical component**

* Understand the principles of server architecture.
* Choose a suitable platform for deploying your IP.
* Deploy your IS on a remote machine (server).
* It is advisable to create container using Docker or Rocker.

The primary means of learning for student is through practice. This is supported and developed through:

1. Guest speakers.
2. Group discussions, reviews and critiques;
3. Working on live projects;
4. Independent study.

**Practice**

The team should select which hosting infrastructure suits their needs (physical hosting or cloud).

The team should argue their decision. This decision should be made based on lawful conditions. The following conditions should be considered:

* storage location (in some cases there is a requirement that clients' information should be stored only locally/in a specific country/not in cloud),
* budget for hosting (in most cases cloud hosting is more expensive than physical hosting),
* performance (CPU/RAM/Disk type (hdd/ssd) /Raid etc.),
* SLA (service level agreement).

**Tasks**

1. SLA (team should decide on the SLA they guarantee their customers).

2. Based on the SLA, the team should determine how many failover regions they should have for their application (as a rule 2-3 areas for mission-critical environments).

3. A team should create an underlying infrastructure plan (in this plan team should show, how many servers and which types of servers they need for their application). For example, it`s a common practice to use servers with SSD hard drives as database servers, because they heavily rely on hard drives.

Team also should select a proper server configuration for their needs. For example, there is no need to have a server with the latest XEON x16 cores for a simple web server with static content.

The team should find a balance between their needs, prices, and configuration.

***Requirement : each IS should have at least 2 DB servers (1 primary and 1 slave), 2 web servers (1 - static content and second for their websites, let`s pretend on PHP or C#), 1 job runner server (as a rule each solution has concept of jobs, job is an executable which executes specific work according to the scheduler (windows scheduler\Crontab\...)),1 load balancer (self-hosted or 3d party service.***

4. Team members should create a little compression matrix of the budget between 1 cloud hosting and one dedicated hosting.

Helpful sources:

* <https://azure.microsoft.com/en-us/pricing/calculator/> - Azure (Cloud)
* <https://calculator.s3.amazonaws.com/index.html> - AWS Cloud
* <https://www.ovh.com/world/dedicated-servers/>(Dedicated) <https://www.soyoustart.com/en/> (Dedicated)
* <https://www.hetzner.com/dedicated-rootserver> (Dedicated).

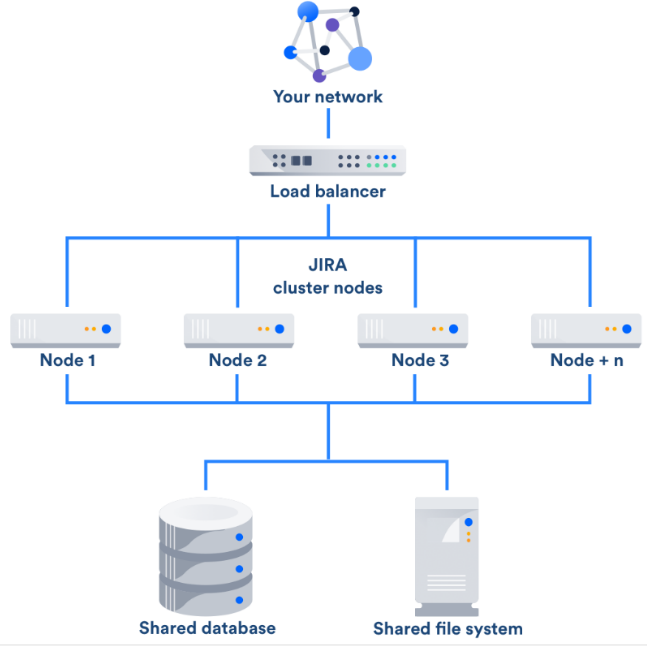
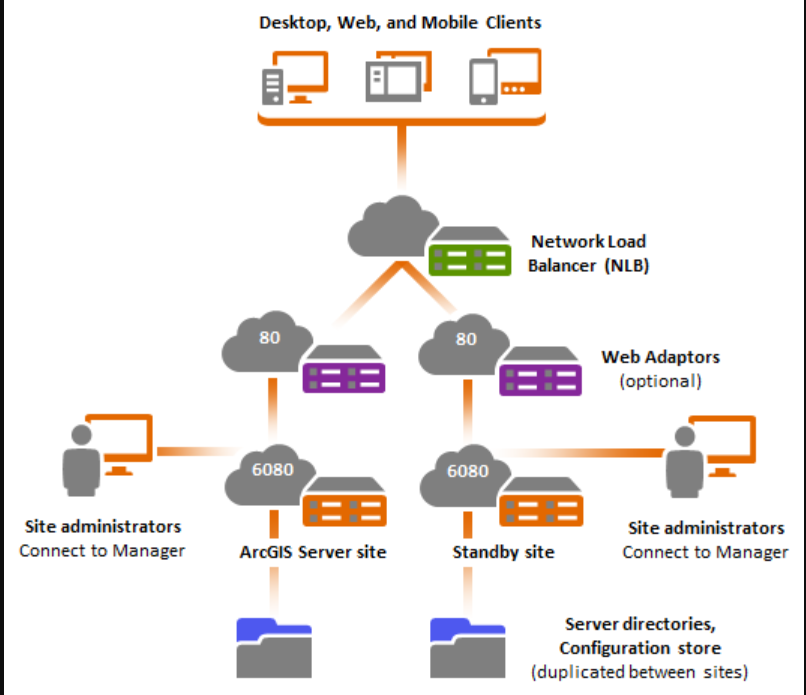
5.The team should create a simple infrastructure map which suits to their needs.

Figure 1. Infrastructure map

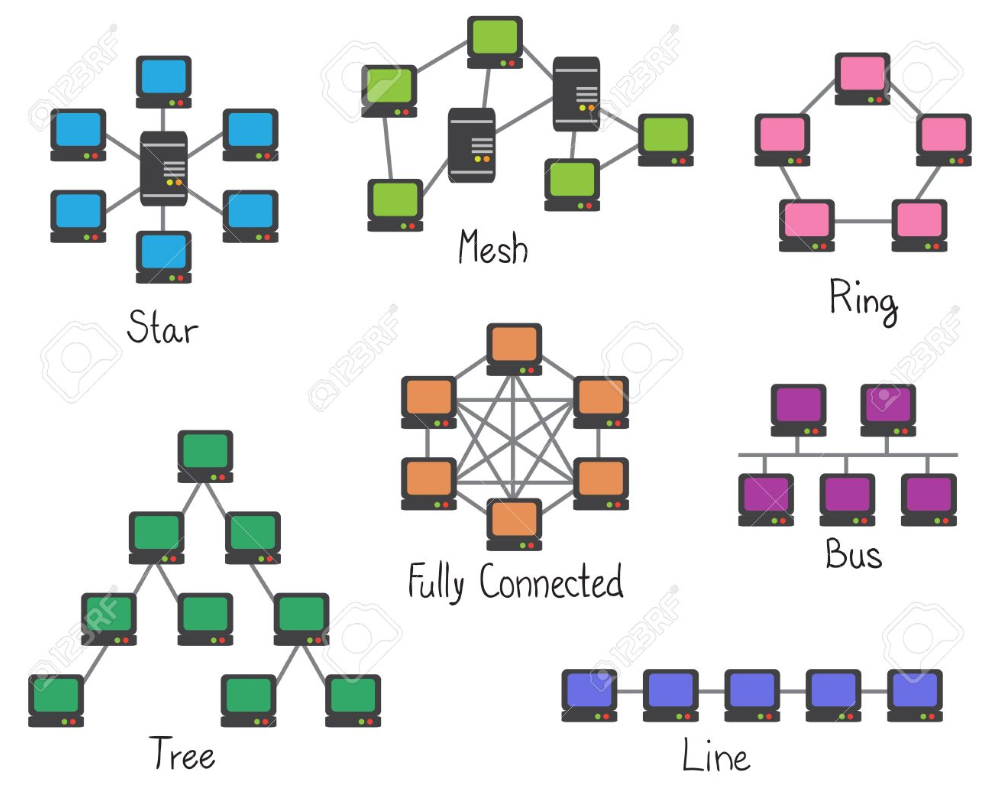
1. The team should decide how they will connect servers inside one region.

Figure 2. Network topology

7. The team should determine how they will connect servers\virtual networks in the different areas. As an example: VPN, GRE tunneling, etc.

***Additional information is available here:*** [***http://www.netheaven.com/TunnelTypes.html***](http://www.netheaven.com/TunnelTypes.html)

**Literature, Courses and Examples**

1. What is SLA? - <https://www.paloaltonetworks.com/cyberpedia/what-is-a-service-level-agreement-sla>
2. Cloud vs hosted hardware - <https://www.networkspecialists.com/the-cloud-vs-the-data-center-whats-the-difference/>
3. What is the difference between data-center and cloud? - <https://www.quora.com/What-is-the-difference-between-data-center-and-cloud>
4. Virtualization - <https://www.vmware.com/solutions/virtualization.html>
5. Proxmox installation guide - <https://www.proxmox.com/en/proxmox-ve/get-started>
6. Vmware install on Windows - <https://www.wikihow.com/Install-Windows-7-on-a-VMware-Workstation>
7. Cloud free accounts - <https://aws.amazon.com/free/>; <https://azure.microsoft.com/en-us/free/>
8. Network tunnels - <http://www.netheaven.com/TunnelTypes.html>
9. What is VPN - <https://en.wikipedia.org/wiki/Virtual_private_network>
10. Simple VPN example - <https://www.softether.org/4-docs/1-manual/7._Installing_SoftEther_VPN_Server>
11. Network mapping tools - <https://www.pcwdld.com/top-10-network-diagram-topology-and-mapping-software>

**Topic 12. Delivery and Deployment**

**Practical component**

* Understand the basic principles of delivery and deployment.
* Develop a continuous integration and delivery process for your IS.
* Choose the optimal version of the virtual machine for your container.
* Add the automated test phase in delivery and deployment process.
* Start the process of automatic creation of production and test version of your IS.
* Add the ability to snapshot and rollback the IS version to the previous one. It is advisable to use TeamCity, Travice or Circle CI.

The primary means of learning for a student is through practice. This is supported and developed through:

1. Project briefings;
2. Real world tasks and examples;
3. Project briefings.

**Practice**

As a rule, there are four main environments in the application lifecycle.

* Dev (developer environment).
* QA (Quality assurance environment).
* Staging (Quality assurance, but with a snapshot from the production database).
* Production (a live environment with live clients).

DevOps should create and manage all these environments. In order to do this automatically (without manual putting files into servers), there is a concept of CD.

Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time, which works together with CI (Continuous integration) principle.

**Tasks**

1. A team should create a CI process which includes the following steps:
   1. Team should agree, which source control instrument works for them (developers + DevOps + managers). For example, basic git flow process which describes branching strategy.
   2. Integration with source control (git\svn\etc.). The primary target is when a new change is pushed into source control (as a rule, release branch\master\dev\ etc. depending on git flow\company processes), a new build should be automatically triggered.
   3. A build should produce artifacts (packages) (zip\tar\nupkg\any other extensions) with a pre-built application, which includes configuration rewrites for a specific environment (if needed). The package should be self-contained, it means that all dependencies should be included in it.
   4. If there are some unit-test projects, these projects should be executed to test, that developers did not break code in a new release.
   5. These artifacts should be available in a feed (for example, Nuget, npm, maven). As a rule, each build server has its own built-in feed)
2. A team should create a continuous delivery flow for their applications:
   1. There should be a deployment flow, which allows distributing packages with the applications to remove servers\infrastructure.
   2. This system should download pre-built packages, which were created by build server and deploy it to infrastructure servers.
   3. As a rule, during a deployment there is a necessity of changing configuration files for an application, for example, based on environment, change connection strings in an application for a database, etc. As a rule, most of the deployment tools have a concept of variables, where DevOps can create variables and reuse them during the deployment process or change in configuration files.

***A team should implement a solution for one of the tasks (not both tasks, only the one task):***

***Task 1***

Let`s pretend that we have a PHP project which is hosted under Linux machine, and you build a CI process which creates a Nuget package with this application and pushes it to the Nuget feed. Now, during deploy to QA\staging\prod, you should apply a number for configuration changes to that application + configuration changes to your web server.

Let`s assume that you have a separate repository and Nuget package with configuration files <https://db.tt/H9yzH0x5ur>. Depending on the environment during deployment, you will need to copy proper configuration file from credentials directory to directory with your applications (for example: (deployment to QA): /credentials/config\_usr.qa.php => /applications/<name>/<version>/config\_usr.php).

Also, you will need to update your web server configuration (because you deployed a new version, for each deployment a folder with new version should be created).

Let`s assume that we are using nginx web server, so depending on the environment your deployment process should change nginx configuration file to a new one, but with updated variables. Here is an example of the configuration with placeholders: <https://db.tt/wDIsTj4x7S>. It means that during the deployment process, your script should replace these variables with the proper value and reload nginx, so it`ll start to server information from a new path.

***When deployment is done, your system should send a simple email to notify, that a new release was deployed to a specific environment. For example:*** [***https://db.tt/nAsw9T417c***](https://db.tt/nAsw9T417c)***.***

***Task 2***

Let`s pretend that you have an ASP.NET project which runs under Windows system. You build a CI process which creates a Nuget package with this application and pushes it to the Nuget feed. Now, during deploy to QA\staging\prod, you should apply a number for configuration changes to that application + configuration changes to your web server. Developers created a system which allows depending on a specific environment variable load different configuration files. Your deployment process should set environment variable that is equal to environment name and deploy a new package into IIS web server.

During deployment to IIS, your script should apply a new path to the application, set application pool identity to "LocalSystem" (<https://db.tt/G5h7txHqQo>) and create a virtual directory with static content (<https://www.dropbox.com/s/n5vw9o3p3ef0va8/1537980345315.png?dl=0>).

***When deployment is done, your system should send a simple email to notify, that a new release was deployed to a specific environment. For example:*** [***https://db.tt/nAsw9T417c***](https://db.tt/nAsw9T417c)***.***

**Tools**

CI: TeamCity, Gitlab, Jenkins, Travis, VSTS, BuildBot, CircleCI, Bamboo, etc.

CD: Jenkins, Gitlab, Travis, AppVeyor, Octopus Deploy, etc.

**Literature, Courses and Examples**

1. CI tools - <https://code-maze.com/top-8-continuous-integration-tools/>
2. CD tools - <https://alternativeto.net/software/octopus-deploy/>
3. A good overview with example of CI\CD process - <https://www.youtube.com/watch?v=JWGLEEm9Qhg>
4. Another overview with example of Gitlab - <https://docs.gitlab.com/ee/ci/examples/>

**Examples**

1. CI\CD with Jenkins - <https://medium.com/@goke/continuous-integration-setting-up-php-applications-in-jenkins-part-1-f6f419a324f7>; <https://modess.io/jenkins-php/>
2. CI\CD with Gitlab - <https://docs.gitlab.com/ee/ci/examples/>
3. CI\CD with TeamCity and Octopus deploy - <https://www.codeproject.com/Articles/851282/CI-CD-using-GIT-Team-City-Octopus>;<https://octopus.com/docs/api-and-integration/teamcity>; <https://octopus.com/teamcity>

**Courses**

1. <https://linuxacademy.com/devops/training/course/name/implementing-a-full-ci-cd-pipeline>

**Topic 13. Migration in IS**

**Practical component**

* Migration of existing infrastructure
* Data migration

Sometimes due to business reasons (as a rule, due to software license limitations, cost or any other), a business can request changes to current IS to cut down the hosting\maintenance costs.

**Practice**

Let`s pretend that currently there is an IS which uses Azure MSSQL database as a primary database, and Amazon S3 storage for file hosting. The business decided to cut down costs for this project, so developer and devops team should migrate database and file storage to another location\system.

Let`s assume that the developer team used ORM system, and changing database for them is only 1 line in configuration (replacing database connection driver), so it`s not a problem from a developers’ perspective. Your team (DevOps), should decide how you can make migration of the current database structure (including all tables, stored procedures, triggers, etc.) and data to another system.

**Tasks**

1.Your team should migrate a database from Azure MSSQL Database to (select one of the following tasks 1.1 or 1.2):

1.1 Self-hosted PostgreSQL database

1.1.1 Using migration tools, your team should create a script\flow which will convert and migrate DB structure from MSSQL database to PostgreSQL

1.2 Self-hosted MSSQL Server

1.2.1 Using Microsoft recommended approach you should migrate a database to a self-hosted solution

2. Amazon S3 storage was removed, you need to create a mechanism which allows to sync files between different servers. The primary idea is that you have a folder and a subset of folders with static resources (images, videos, etc.). You should create a mechanism, where you have one primary server with these files, and when you add\remove\change files on that server, these changes should be applied to all other servers using some sync mechanism.

**Literature, Examples:**

1. PostgreSQL migration tools - <https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL>
2. One more guide MSSQL => PostgreSQL - <https://www.devbridge.com/articles/migrating-from-mssql-to-postgresql/>
3. MSSQL to PostgreSQL guide - <https://wiki.postgresql.org/wiki/Microsoft_SQL_Server_to_PostgreSQL_Migration_by_Ian_Harding>
4. AzureSQL => SelfHosted sql - <https://www.magnetismsolutions.com/blog/dinesh's-blog/2015/04/13/move-sql-azure-database-to-local-sql-server>
5. AzureSQL => Selfhosted 2 <https://docs.microsoft.com/en-us/dynamics365/unified-operations/dev-itpro/database/copy-database-from-azure-sql-to-sql-server>
6. Rsync sync - <https://en.wikipedia.org/wiki/Rsync>
7. Sync tools and description - <https://superuser.com/a/703606>

**Topic 14. IS Support and Maintenance**

**Practical component**

* Hardware performance monitoring
* Availability monitoring
* Backup and recovery of IS
* On-site support and consultancy

**Objective**

1. Setup and configure availability and performance monitoring for IS.

2. Setup backup system for IS and IS database.

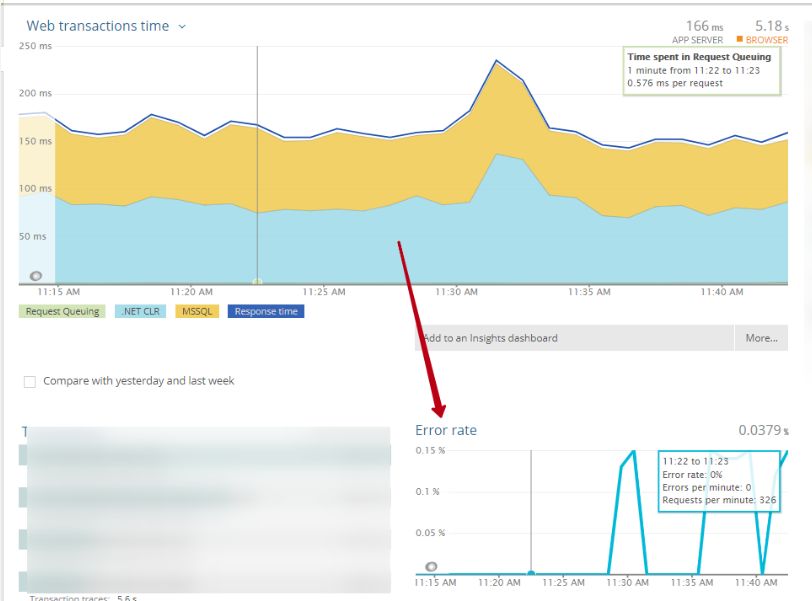
**Tasks**

1. DevOps team should create "monitors" which should monitor IS availability from different locations (let`s pretend that IS contains a public website). (ref *Availability monitoring tools*)

Figure 3. HTML monitoring

* 1. IS should be monitored at least from three different geolocations.
  2. If at least two monitors fail, an email notification should be sent to the DevOps team

1. On each server, DevOps team should set up some "logs and data collectors" which will push information to the monitoring system.
   1. DevOps team should create some basic alerts policies:
      1. Server is not responsible (not pushing logs) for 3 minutes
      2. CPU load is more than 70% for 5 minutes
      3. CPU load is less than 10% for 3 minutes
      4. RAM usage is more than 60% for 5 minutes
      5. HDD free space is less than 30%
      6. HDD ActiveTime is more than 90% for 3 minutes
2. Application monitoring (ref *Performance monitoring tools*).
   1. DevOps team should create some basic alerts policies for IS performance monitoring:
      1. Application Error rate > 5 %

Figure 4. Monitoring dashboard 

3.1.2 Abnormal execution time\spikes for the application

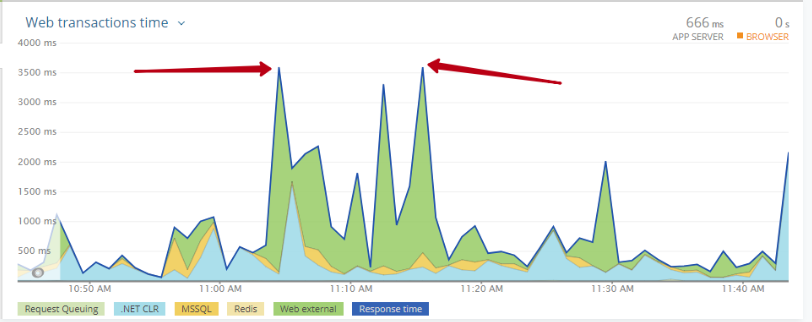


Figure 5. Monitoring chart

1. Based on the IS technology stack DevOps team should decide which backup technologies they can use to perform system backup and restore efficiently.
   1. First of all, DevOps team should decide, where they will store backup data (in local DC, in remote file storage, etc.).
   2. Each backup implementation for this task should consist of two parts: database backup and VM\data backup (Please select one of the following tasks 4.2.1 or 4.2.2 based on your IS stack).
      1. **Linux & PostgreSQL** 
         1. Team should create a script which will create full database backup every 10 hours
         2. Team should create a script which will create a differential backup every 3 hours
         3. Team should create a script which will create complete VM (Virtual machine) or complete server backup every three days
      2. **Windows & MSSQL** 
         1. Team should create a script which will produce full database backup every 10 hours
         2. Team should create a script which will create a differential backup every 3 hours
         3. Team should create a script which will generate complete VM (Virtual machine) or full server backup every three days
2. Let`s pretend that we have QA and Staging environment for our IS. As a rule, staging database should be replaced by production database every week. DevOps team should write a script which will be started by crontab\windows task planner each week on Sunday at 10 PM; this script should download latest available backup for database and automatically refresh QA and Staging database with data and structure from this backup.

**Literature and reference**

1. Availability monitoring tools - <https://www.pingdom.com/>;<https://uptimerobot.com/>; <https://www.zabbix.com/>
2. Performance monitoring tools - [https://www.site24x7.com/](https://newrelic.com/%20https://www.site24x7.com/);<https://www.datadoghq.com/>; <https://www.zabbix.com/> ; <https://newrelic.com/>
3. Newrelic docs - <https://docs.newrelic.com/>
4. Backups : Postgre - <https://wiki.postgresql.org/wiki/Automated_Backup_on_Linux>
5. MySQL - <https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/create-a-full-database-backup-sql-server?view=sql-server-2017>
6. RedGate toolbelt - <https://www.red-gate.com/products/sql-development/sql-toolbelt/index>
7. Linux backup - <https://help.ubuntu.com/lts/serverguide/backup-shellscripts.html.en>
8. Newrelic alrets guide - <https://docs.newrelic.com/docs/alerts/new-relic-alerts>

**Examples**

1. Policies: <https://db.tt/ZvwgDVcLyX> <http://blog.newrelic.com/wp-content/uploads/AlertPolicy.png>
2. <https://blog.newrelic.com/product-news/reintroducing-windows-server-monitoring/>
3. <https://www.zabbix.com/forum/zabbix-help/50963-monitoring-cpu-usage-on-windows>