Deployments, Operations

Jakub Danek
Yoso Czech s.r.o.

mailto:jakub.danek@yoso.fi
https://www.linkedin.com/in/danekja/

Motivation

- **Deployment significantly influences application´s behaviour**
  - Security
  - Performance
  - Downtime during upgrades
- **Current trend is to minimize distance between development and operations → DevOps**

Topics

- What types of environments we have
- How can we deploy applications?
- Who is actually responsible for application deployments?
- Operations – backups, monitoring
- Examples – load-balanced and high-availability environments

Environments

- **Several different environments can be used during development cycle**
- **Each has different requirements on**
  - Hardware (performance)
  - Security
  - Stability
- **Each has different purpose**

Environments - Development

- **Used by developers to test the application outside of their workstation**

- **Performance requirements: low**
- **Stability requirements: low**
  - → developers can update the deployment several times a day, as much as needed
- **Security requirements: low**
  - No real data

Environments - Testing

- **Used by testers/product owners to validate/verify the implementation**

- **Performance requirements: low-medium**
  - Needs to respond in reasonable time, but very few users
- **Stability requirements: medium**
  - Testers/product owner need time to test new deployments, shouldn‘t change without notice
  - Sudden downtime does not cause any serious issues though
- **Security requirements: medium**
  - No real data, but setup should be production-like as much as possible

Environments - Pre-production

- **Used by testers/operations to test deployment in production-like environment, performance testing, security testing etc.**

- **Performance requirements: high**
  - Should copy production environment as much as possible
- **Stability requirements: med-high**
  - Tests deployment in production-like environment → deployments should happen based on predetermined schedule
  - Any issues causing downtime are a serious threat at this stage of development
- **Security requirements: high**
  - No real data, but same requirements as in production environment

Environments - Production

- **Used by customer**

- **Performance requirements: high**
  - Performance issues cause unhappy users
- **Stability requirements: high**
  - Deployments happen only at predetermined times
  - Users do not experience any downtime vs. down-time must be well communicated to users
- **Security requirements: high**
  - Real data, security vulnerabilities represent a serious threat

Environments - Other

- **Special testing environments can exist as well, e.g.:**
  - Performance testing
    - Maintaining production-like setup for testing can be expensive
      - → special cloud-based environment started only during the test-runs
  - Big feature test-env
    - Features taking long time to develop
    - Maybe large overhaul of the application is needed
    - → separate testing environment so that main development is not disrupted
    - → removed after the feature is finished and merged into main codebase

Naive deployment

- All done manually – build, deployment, configuration…
- No extra infrastructure needed
- Prone to human errors
- Slow

Example Process: deployment_naive.pdf

Scripted deployment

- **Improvement over naive approach – as much as possible automated with (shell) scripts**
  - Upload to server
  - Applying proper configuration
  - Restarting the service
  - Testing
- **But, developer is still required to exec those scripts on the server**

Example Process: deployment_scripted.pdf

Automated deployment

- **All steps fully automated**
- **Designated build server**
  - Application responsible for all steps related to deployment
  - Deployment is triggered by clicking single button, after that no action is needed
  - Minimal human input → all deployments go through same process

Example Process: deployment_automated.pdf

Continous Integration

- **Practice aiming at integrating new changes into main codebase as soon as possible**
  - Limits overhead of integrating own changes and changes done by other team-members
  - Requires high-quality testing process to ensure code stability
    - → build server can be configured to build and run all tests after every commit to master
    - → potential errors are discovered quickly
    - → developers can continue with their work while build server runs (timely) tests

Continous Delivery

- **Practice aiming at deploying changes in small cycles**
  - A flexible approach to delivering new features/fixing critical errors
  - Possible due to small deployment overhead when using build server
- **Whole team needs to see whole build and deployment process (build, tests, reports) → easier to react**
- **Due to full automation, any version can be deployed to any environment at any time**
  - With constraints of course (e.g. you need to ensure no downtime of service during deployment to prod)

Build Servers

- **Jenkins**
  - Open-source, widely used, lots of plugins
- **TeamCity**
  - JetBrains
- **Travis CI**
  - CI server for GitHub hosted projects
- **Team Foundation Server**
  - Microsoft

Who is responsible?

- **Operations team (system administrators)**
  - Understand the servers, configuration
  - Lack detailed knowledge of the application
- **Developers**
  - Often do not want to deal with system administration
  - Have detailed knowledge of the application

DevOps

- **An approach/movement narrowing the gap between development and operations**
- **Developers and sysadmins need to cooperate**
  - Teams should be put together so that each team is able to
    - Deliver functionality
    - Test
    - Deploy
    - Maintain

DevOps

- **Automation is crucial**
  - Limit manual work
  - Less error-prone
  - Fast delivery
- **Build and deployment automation**
  - We have just discussed that

DevOps

- **Infrastructure automation**
  - DevOps promotes same principles for infrastrucutre management as developers use
  - Describe your infrastructure in configuration files
    - Can be versioned
    - Configuration management tools (Ansible, Puppet, Chef)

DevOps

- **Virtualization**
  - Cloud allows easy scaling and orchestration of new servers
  - Virtualization tools/providers allow easy creation and management of identical server instances
  - AWS, OpenStack, vagrant…
- **Containers**
  - Another level of virtualization – allow shipping application together with its full runtime environment
  - Build once, run anywhere
  - Docker

DevOps

- https://theagileadmin.com/what-is-devops/
- https://www.atlassian.com/devops

Other Operations Tasks

- Security Configuration
- Monitoring

  + scaling
- Backups

Security

- **It is common to deal with the following security tasks**
  - Firewall configurations
  - Certificate management
    - HTTPS configuration

Security

- **Firewall configuration**
  - Allow connection only where needed, for example:

    Application Server needs connection from internet on 80/443 ports (HTTP)

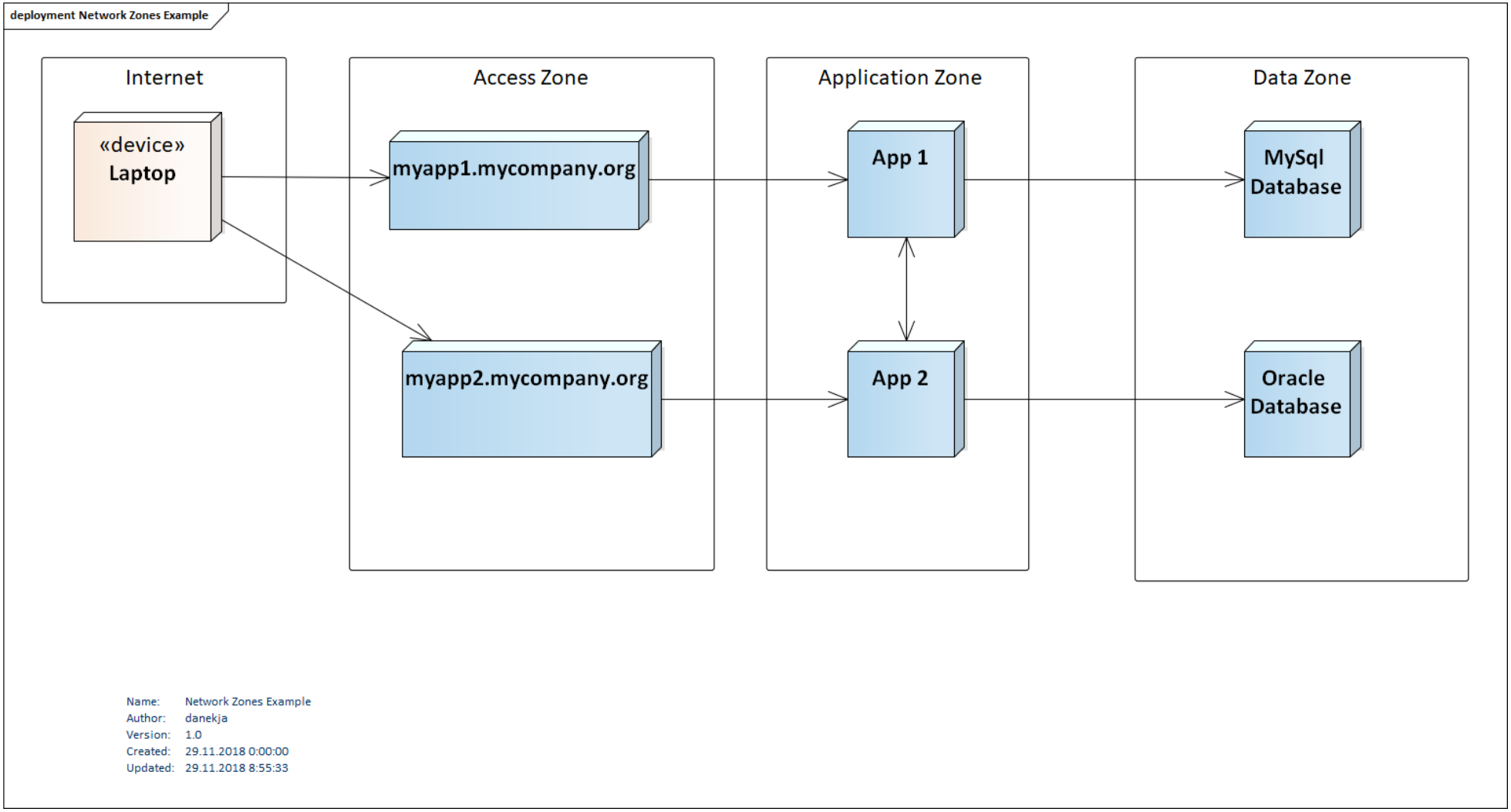
    Database Server needs connection from the application server on 3306 port (MySQL)

    Both need connection on 22 port (SSH) from company´s internal network.

Security

- **Firewall configuration**
  - Companies with many servers tend to group them into network zones based on purpose
  - → certain zones are unreachable from the internet
    - Implemented by firewall at entry point to the network
    - Useful for networks holding data stores (databases, file storage…)
  - → it is necessary to configure proxies which have been granted access to communcate between zones
  - → only a few points of access in space of hundreds of servers

Security – Network Zones Example



Security

Certificate Management

- **Each certificate consists of public (certificate itself) and private (key) parts**
- **Public part has usually validity period (quite often one or two years)**
  - After that it needs to be renewed

Security

Certificate Management – New Certificate

- **Private key is generated as random data**
  - Can be protected by passphrase, but not recommended for web certificates (you would need to solve the problem of providing the password each time web-server is restarted)
- **Using the private key, Certificate Signing Request (CSR) is made**
- **CSR is sent to certificate authority (CA) for signing**
  - CA verifies you own the domain for which you request certificate
- **Certificate file is received**

Security

Certificate Management – Certificate Structure

- **Public Certificates defined in X.509 standard**
- **When creating CSR, several fields are asked:**
  - CN = Common Name = Fully Qualified Domain Name of your server
    - e.g. myapp1.mycompany.org
    - Browsers will reject the certificate if CN does not match the domain from which the certificate came
  - Country code, State, Organization, Organization Unit, email
    - Important for end-user to get information about your company, technically not necessary

https://www.itu.int/rec/T-REC-X.509/en

Security

Server Configuration

- Web server needs access to the files with public and private parts

Example: nginx

```
server {
    listen              443 ssl;
    server_name         www.example.com;
    ssl_certificate     /etc/pki/tls/certs/www.example.com.crt;
    ssl_certificate_key /etc/pki/tls/private/www.example.com.key;
    ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers         HIGH:!aNULL:!MD5;
    ...
}
```

Src: nginx.org

Security

Questions?

Monitoring

- **You require information in order to react/prevent operation problems**
  - How much RAM has the server left?
  - How much free disk space there is?
  - Is the server under extensive load?
  - Does the web application answer to requests?

Monitoring

- **Checking manually when problems occur is too late!**
  - → monitoring application
- **External**
  - No special software required on the servers
  - Can monitor things like:
    - Application can be reached via network and answers requests
    - Response times of the application are within limits
    - Validity of HTTPS certificates

Monitoring

- **In-place**
  - Monitoring software daemon installed on the server
  - Can monitor things like:
    - CPU load
    - Memory use
    - Remaining free disk space

Monitoring

- **All inputs from all servers aggregated at one place**
- **Dashboards with status**
- **Email/SMS/Chat messages if something goes wrong**
  - If you realize your server is running out of disk space, you can react before users experience any issues
- **Support for maintenance windows (no alarms are raised if you inform the monitor that downtime is due to planned maintenance)**

Monitoring

Site24x7

Search Monitors/Groups/Tags

Home
Web
APM
Server
VMware
AWS

Monitor status

Last updated **a few seconds ago**

Monitor Name	Performance	Last Polled
brafitter-sizequery Lambda Function |	0 Count	4 minutes ago
brafitter-update Lambda Function |	0 Count	2 minutes ago
dynamodb SNSTopic |	-	5 minutes ago
Membernet demo Website |	192 ms	2 minutes ago

Monitoring

https://www.nagios.org/

https://icinga.com

https://www.site24x7.com/

www.datadoghq.com

https://prometheus.io/

and many more…

Backups (and restores)

- **Data are the most valuable thing**
- **→ any production system needs to have backup mechanism**
- **In case of error data can be restored without too many losses**
  - Attack
  - Failed upgrade with broken data migration
  - SW/HW failure
  - User error

Backups (and restores)

- **Backup properties**
  - Taking backup must not influence system runtime
  - Can restore system to fully consistent state
  - Stored at separate location
- **Full vs Incremental**
  - Backup vs restore speed
  - Space vs consistency threat

Examples

Examples – load-balanced environment

Nginx:

```
http {
    upstream myapp {
        server myapp1.intranet.mycompany.org;
        server myapp2.intranet.mycompany.org;
        server myapp3.intranet.mycompany.org;
    }

    server {
        listen 443 ssl;
        ...
        location / {
            proxy_pass http://myapp;
        }
    }
}
```

Load-BalancedEnvironment.pdf

Examples – load-balanced environment

HA Proxy:

```
frontend http
  bind myapp.mycompany.org:443 ssl crt /etc/pki/tls/certs/myapp.pem
  mode http
  default_backend myapp

backend myapp
   balance roundrobin
   mode http
   cookie MYAPPLBSRV insert indirect nocache maxidle 4h
   server myapp1 myapp1.intranet.mycompany.org:80 check cookie myapp1
   server myapp2 myapp2.intranet.mycompany.org:80 check cookie myapp2
   server myapp3 myapp3.intranet.mycompany.org:80 check cookie myapp3
```

Load-BalancedEnvironment.pdf

Examples – high-availability environment

- **Load-balancer solves two problems**
  - Performance: Users can be distributed among multiple application nodes
  - Stability: if one node dies, users automaticaly reach another, via same URL
- **What if load-balancer dies?**
  - Virtual Router Redundancy Protocol (VRRP)

Examples – high-availability environment

- **Virtual Router Redundancy Protocol (VRRP)**
  - Creates Virtual Routers – group of multiple real routers
    - 1 active, rest backup
  - Routers within group communicate with each other
    - → if active router dies, new active is elected from the backups
- **How can we use this on application layer?**
  - Keepalived (http://www.keepalived.org/)

Examples – high-availability environment

- **Keepalived – need to configure the following on each node**
  - Virtual IP (address to which clients connect)
    - Always resolves to the active node
  - Virtual Router ID (unique VRRP group identifier within the same network segment, 0..255)
  - Check script/command
    - Keepalived calls this command regularly to check if the relevant service is running
      - In our example it would check e.g. if haproxy process is running
  - Priority: used during master elections
  - Password: used to secure communications between the instances

High-AvailabilityEnvironment.pdf