IIT Madras
BSc Degree

Deployment

Deploying an App

- Components of an App
- Service approaches
- Automation and Containers

Developing an App

- Idea

Developing an App

- Idea
- Local development
  - File system
  - Editors, Desktop, Documents, File management

Developing an App

- Idea
- Local development
  - File system
  - Editors, Desktop, Documents, File management
- Single computer

Developing an App

- Idea
- Local development
  - File system
  - Editors, Desktop, Documents, File management
- Single computer
- Multiple services
  - Web server
  - Database server

Permanent Deployment

Permanent Deployment

- Dedicated servers

Permanent Deployment

- Dedicated servers
- Always-on internet connection

Permanent Deployment

- Dedicated servers
- Always-on internet connection
- Uninterrupted power

Permanent Deployment

- Dedicated servers
- Always-on internet connection
- Uninterrupted power

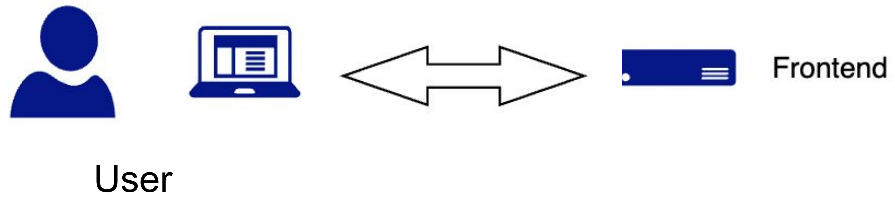
Infrastructure!

- Data Centers

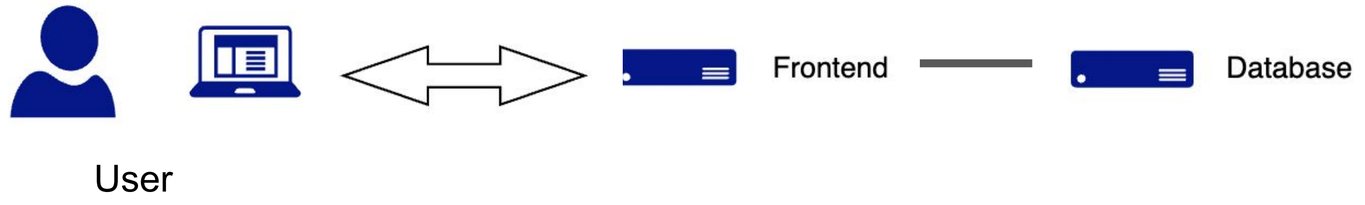
Cloud

Scaling

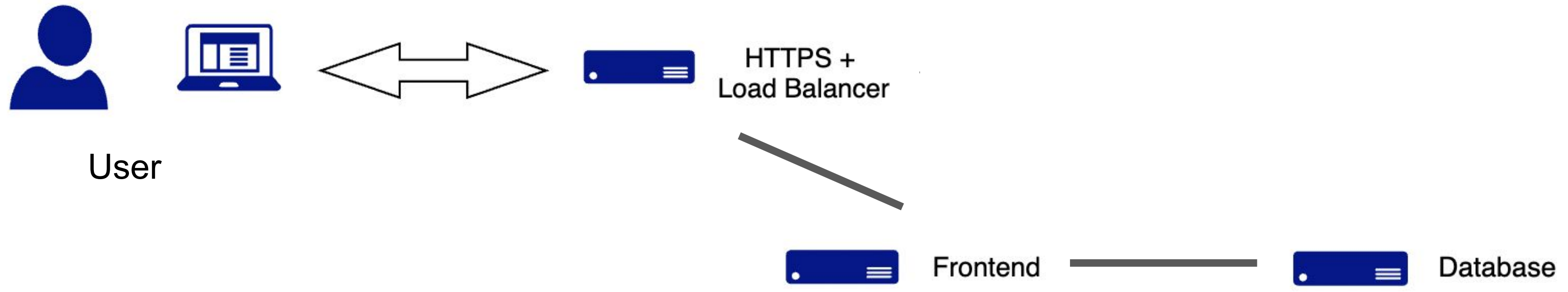
Scaling



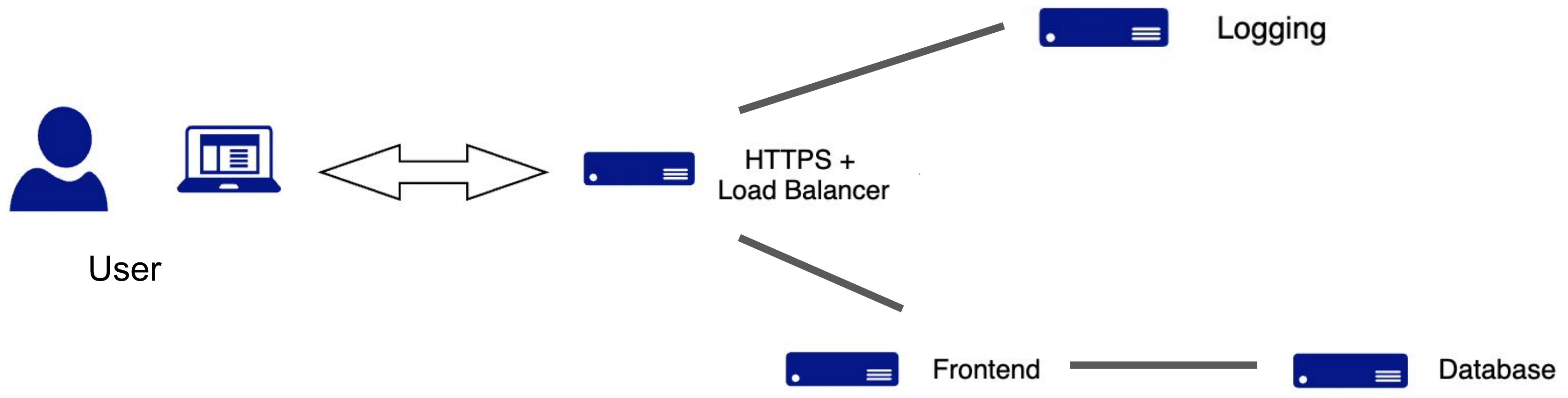
Scaling



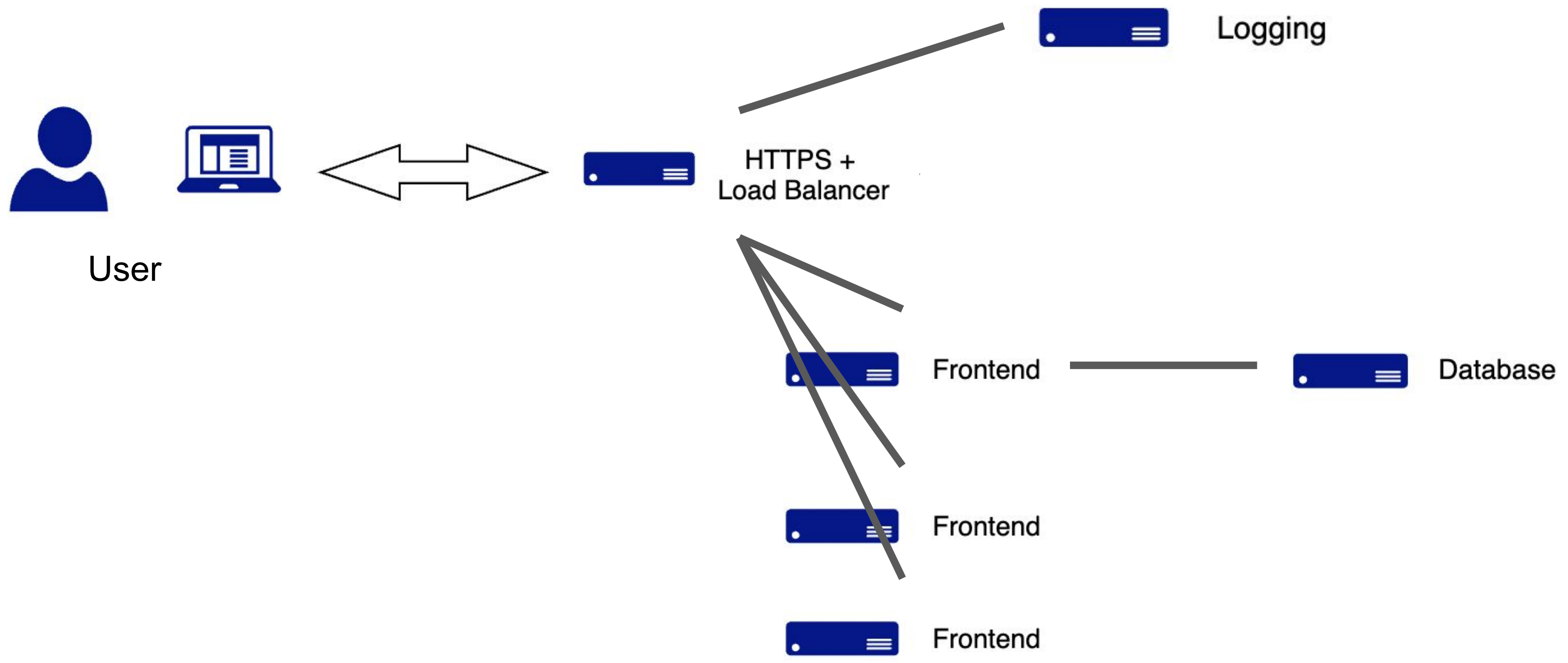
Scaling



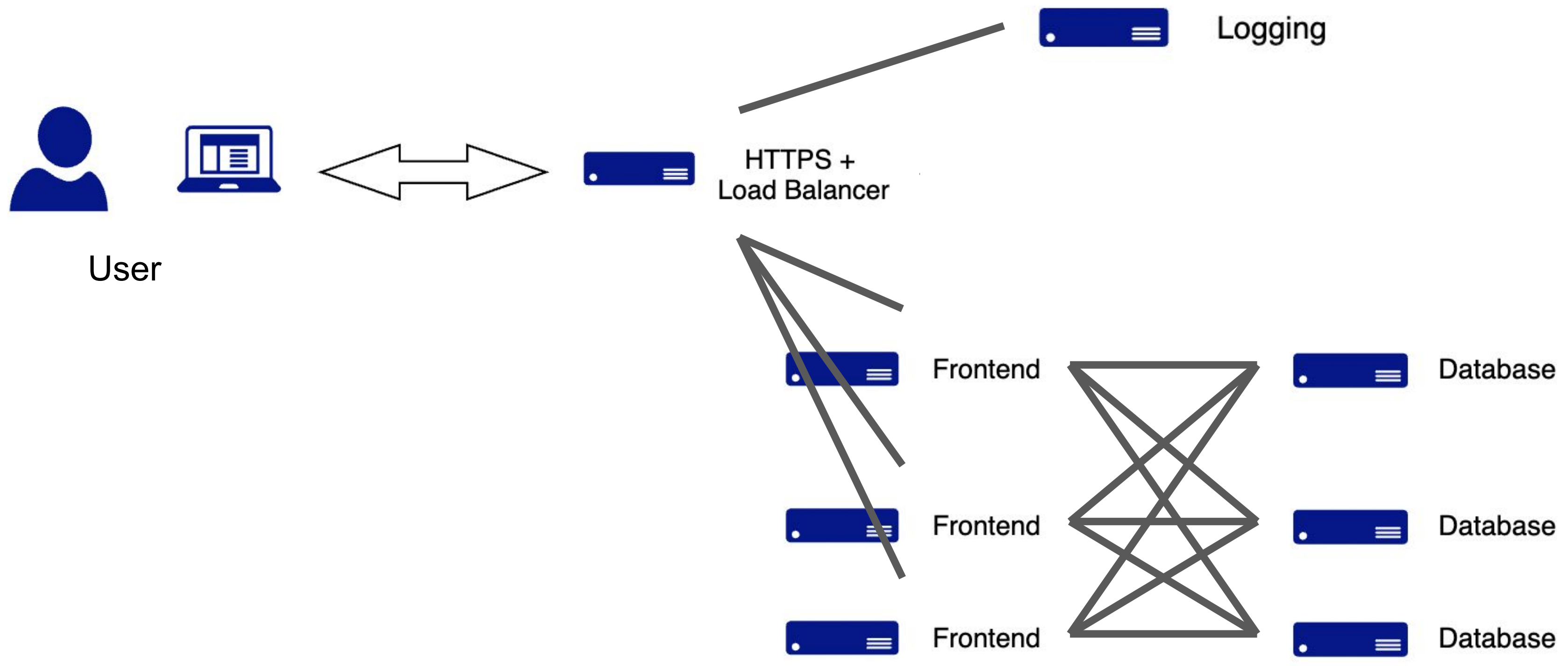
Scaling



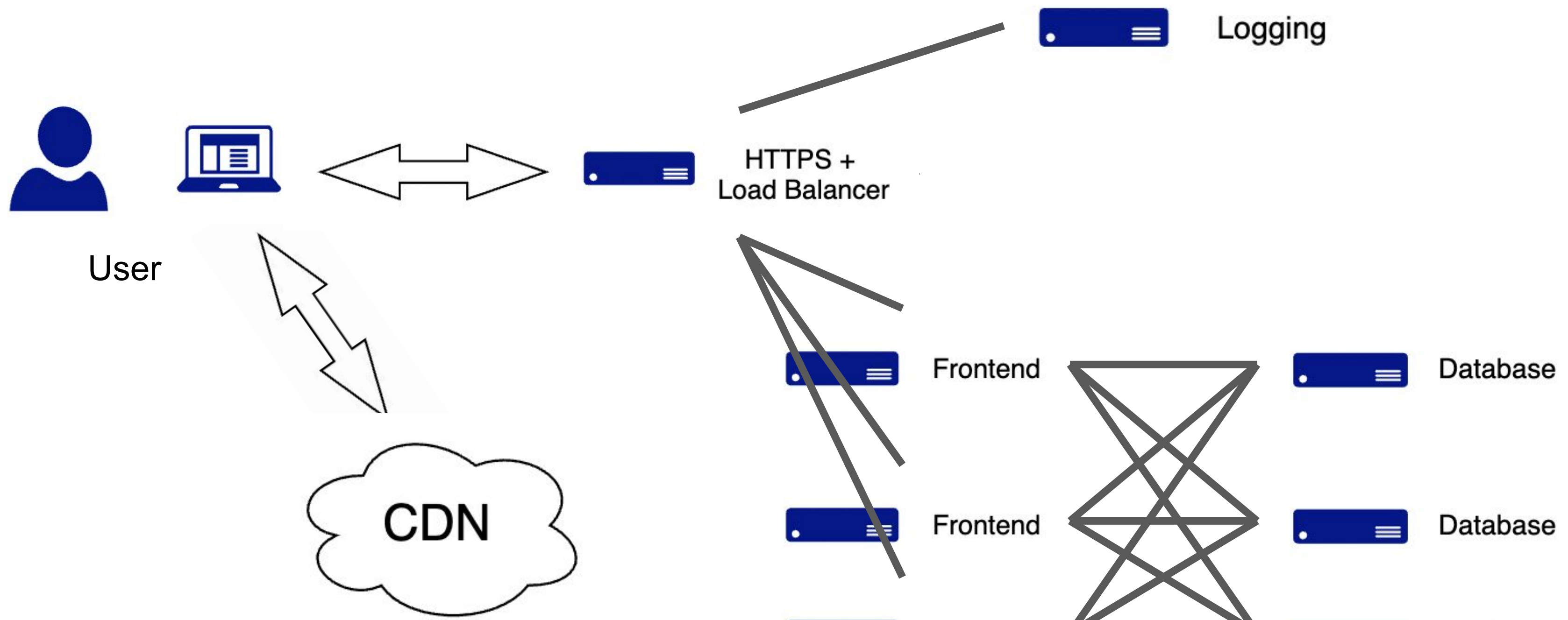
Scaling


Logging
User
HTTPS +
Load Balancer
Frontend
Database
Frontend
Frontend

Scaling


Logging
User
HTTPS +
Load Balancer
Frontend
Frontend
Frontend
Database
Database
Database

Scaling


Logging
User
HTTPS +
Load Balancer
CDN
Frontend
Frontend
Frontend
Database
Database
Database

Summary

- Base development of an app is easy
- Deployment is hard!
- Infrastructure
  - Always on servers - auto restart
  - Always on network
  - Uninterrupted power
  - Monitoring and logging

Services Approach

- SaaS
- IaaS
- PaaS

Service approach

- Specialization
- Datacenter operators specialize in infrastructure
- Developers focus on app development
- Standard software deployments?

Software-as-a-Service (SaaS)

- Online office platforms
  - Google docs, spreadsheets, Office 365
- Content Management Systems
  - Drupal, Wordpress
- Issue tracking
  - Trello, Redmine

Hosted solutions: all the software is installed and maintained

Infrastructure-as-a-Service (IaaS)

- Raw machines (or virtual machines)
- Power, networking taken care of
- Install your own OS
    - Manage OS upgrades, security patches, software updates

Cloud compute systems:

- AWS
- Google Compute Engine
- Azure
- DigitalOcean, Linode, ...

Platforms

- Combination of hardware and software
- Specific hardware requirements
  - Computing power, RAM, disk
- Specific software requirements
  - OS version, automated updates and security, firewalls
- Custom application code
  - Flask, RoR, Laravel, ...

Platform-as-a-Service

- Provider takes care of:
  - Power, network, machine management
  - OS installation, security patches
  - Base application platform: Python+Flask, PHP+Laravel: maintain multiple versions, manage security updates
  - Multiple databases and connectivity options
- Developer needs to:
  - Manage application code
  - Specify requirements on server sizing, database, connectivity
- Scaling
  - Combined inputs from developer and provider

Examples

- Replit: https://replit.com/@nchandra/flasktest#main.py
- Glitch: https://glitch.com/edit/#!/gusty-sage-constellation?path=server.py%3A1%3A0
- GAE: https://flasktest-328815.uc.r.appspot.com/
- https://shell.cloud.google.com/?page=editor&show=ide%2Cterminal
-

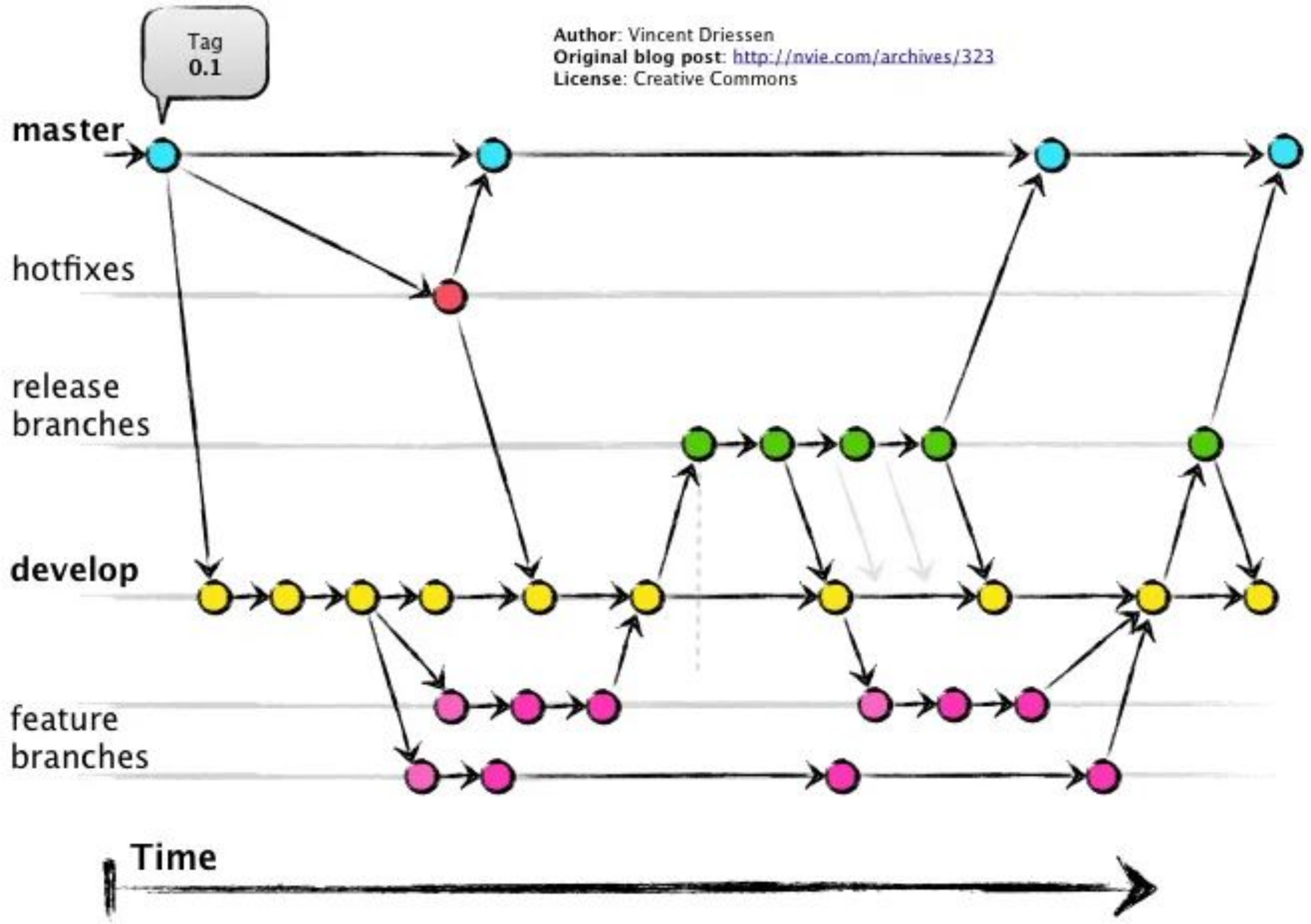
Summary

- PaaS: provide platforms to build on
    - developers focus on code
- Varying degrees of complexity, ease of use
    - Replit, Glitch - GAE, AWS ElasticBeanStalk, Heroku
- Integrate with other code development practices:
    - version control
    - continuous integration (testing)
    - continuous deployment
    - scaling and automation

Deployment

Version control

- How to manage changes to code?
- Retain backups of old code
- Develop new features
- Fix bugs



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons

Version control

- Centralized
  - central server, many clients
  - push changes to server each time
  - multiple editors? Lock files? Merge?
- Distributed
  - can have central server but not needed
  - changes managed using “patches” - email, merge requests, …
- github, gitlab etc.
  - centralized on top of distributed
  - friendly interfaces
  - worth learning command line

Continuous integration

“practice of automating the integration of code changes from multiple contributors into a single software project”

- Atlassian documentation

CI workflow

- Integrate with version control
- Multiple authors contribute to different parts of code
- Central “build server” automatically compiles/builds code

Automation is the key here

Best practices

- Test driven development
    - Write tests before code
- Code review
    - Pull and merge requests - enabled by web interfaces like github/gitlab
    - Review code for correctness, cleanliness, style, …
- Integration pipeline optimization
    - Tests run on each push to server - can be several times a day
    - Fast runs, optimized based on changes etc.

Continuous Delivery / Deployment

- CI/CD - parts of “DevOps” pipeline
- CI = Continuous Integration
- CD could be
  - Continuous Delivery
  - Continuous Deployment

Continuous Delivery

- Once CI (testing) passed, package files for release
- Automated delivery of “release package” on each successful test
- Why?
    - Nightly builds
    - Beta testing
    - Up-to-date code version

Continuous Deployment

- Extend beyond Delivery: Deploy to production
- Passed tests -> deployed to users
  - Users see latest version that has passed tests
  - No installing new versions / updating code or servers
- Benefits
  - Immediate fixes, upgrades
  - Latest features deployed immediately
- Drawbacks
  - Tests may not catch all problems!

Containers

- What?
  - self-contained environment with OS and minimal libraries - just enough to run process
  - Primarily used with Linux kernel namespaces, others like chroot possible
- Why?
  - Full OS impossible to version control - too much software, too many versions
  - Create self-contained images that can be version controlled
  - Sandboxing - image cannot affect other processes on system
- How?
  - Kernel level support needed
  - All communication “inter-container” - networking

Containers

- chroot
    - custom filesystem for part of the code
    - no real process isolation
- FreeBSD jails, Linux VServer, OpenVZ
    - containers in Linux - same kernel, different filesystems
- Control Group namespaces (cgroups) - Linux kernel 2008
    - process isolation through namespaces
- docker
    - mechanisms for managing images - popularized containers
    - problems: bad practices, version control difficult etc.

Orchestration

- App consists of multiple processes, not just one
- Start in some specific order (dependencies)
- Communicate between processes that are isolated
    - Network
- Mechanisms to build and orchestrate, automate
    - docker-compose
    - Kubernetes
- Key to understanding and managing large scale deployments

Summary

- App: idea to deployment
  - Requirements - Tests - Code - Integration - Delivery - Deployment
  - Scaling
- Mechanisms
  - HTML + CSS + JS - Frontend user interface
  - Databases, NoSQL, cloud stores - Backend
  - Authentication, proxying, load balancing - “middleware”
  - Platform-as-a-Service - deployment and change management