Universidad de Oviedo

EN
English

School of Computer Science

Allocation

SOFTWARE ARCHITECTURE

Course 2019/2020

Jose E. Labra Gayo

Allocation

Relationship between Software and its execution environment

Where does each component run?

Allocation

- Deployment view
- Packaging, distribution, deployment
  - Distribution channels
  - Delivery options
  - Execution environments
  - Deployment pipeline
- Software in production
  - Configuration
  - Capacity planning
  - Logging & Monitoring
  - Incidents & post-mortem
  - Chaos engineering

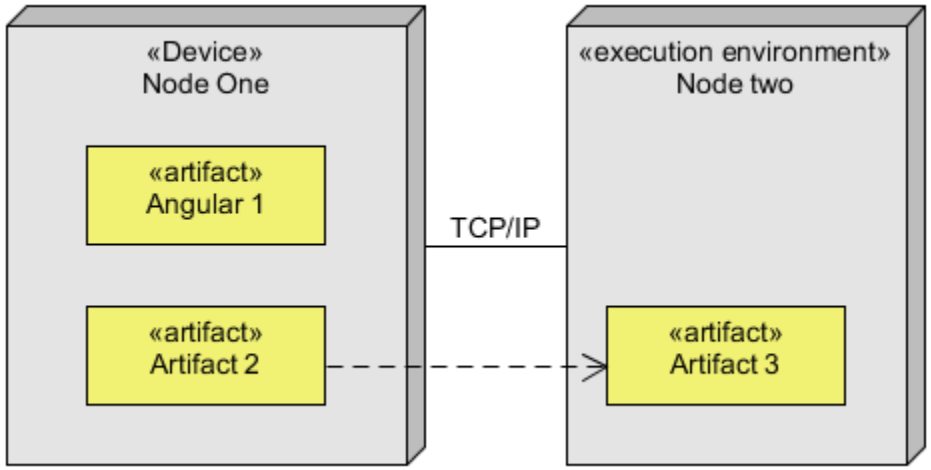
Deployment view

UML has deployment diagrams

Artifacts associated with computational nodes

2 types of nodes:

- Device node
- Execution environment node



Packaging, distribution & deployment

Packaging

Create an executable from source code

Package consists of:

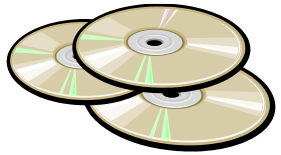
- Compiled code
  - Even for interpreted languages:
    - Transpiled, obfuscated & minimized
- Configuration files
  - Environment variables
  - Credentials, etc.
- Libraries & dependencies
- Installation scripts
- User manuals & docs

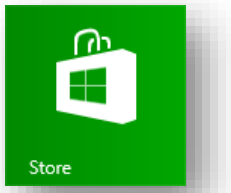
Publishing releases

- A *release* implies functionality changes
- Planning
  - Publishing a release has costs
    - Usually, current users don't want new releases
  - External factors:
    - Marketing, clients, hardware, ...
  - Agile model: frequent *releases*
    - Continuous delivery minimizes risk

Distribution channels

- Traditional distribution
  - CDs, DVDs, ...
- Web based
  - Downloads, FTP, ...
- Application markets
  - Linux packages
  - App stores:
    - AppStore,
    - Google Play,
    - Windows Store





Software computation options

- Data Center (On-premises):
  - Se instala y ejecuta en computadores del cliente
- Cloud computing: SaaS (Software as a Service)
  - Computer resources on demand
- Edge computing
  - Computation done near customer devices
    - Connected devices process data closer to where it is created
    - Example: IOTs, Connected cars, ...
- Fog computing
  - Computation at intermediate nodes (Local Area Network)

Software computation options

Cloud layer

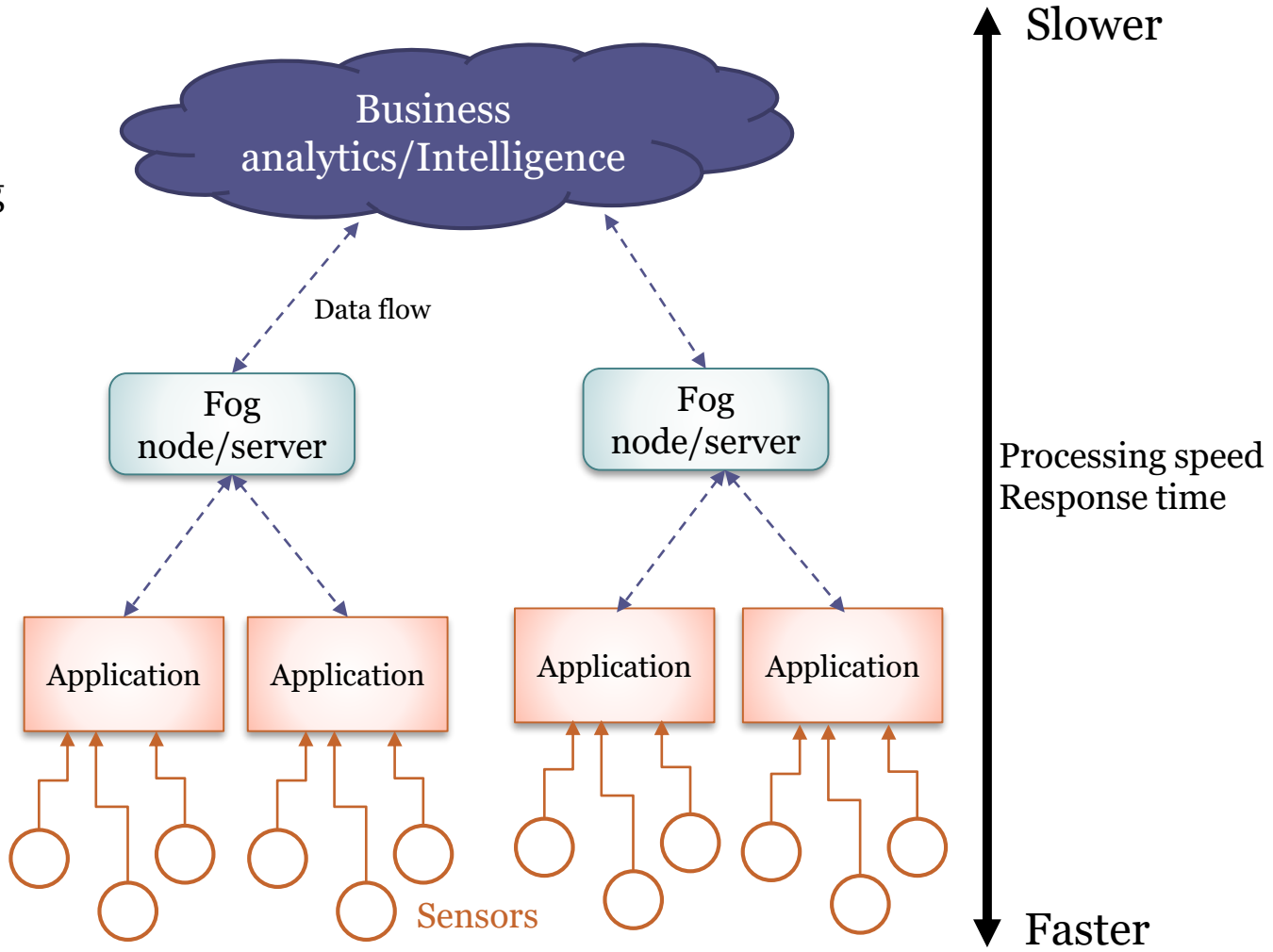
Data centers
Big data processing
Data warehousing

Fog layer

Local network
Control response

Edge layer

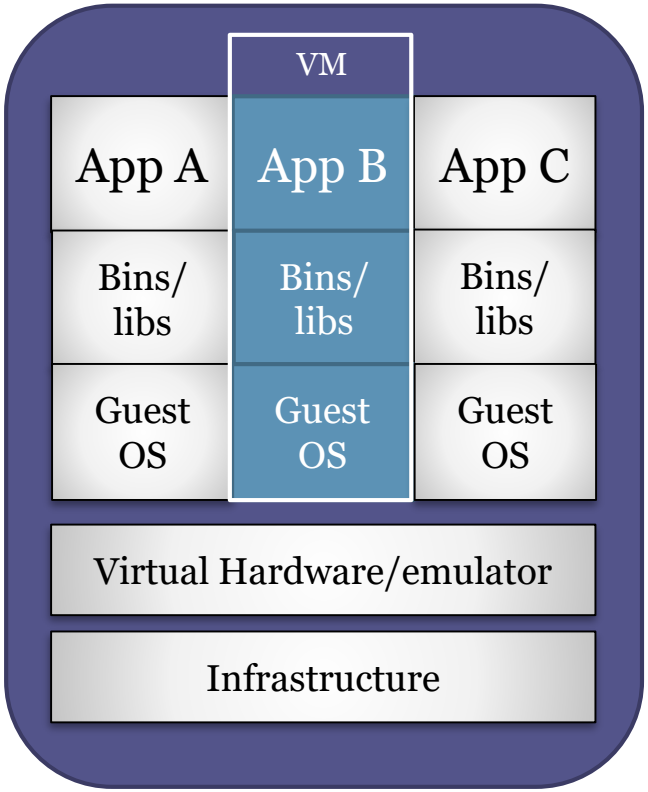
Real time
Micro data storage
On-premises visualization
Embedded systems

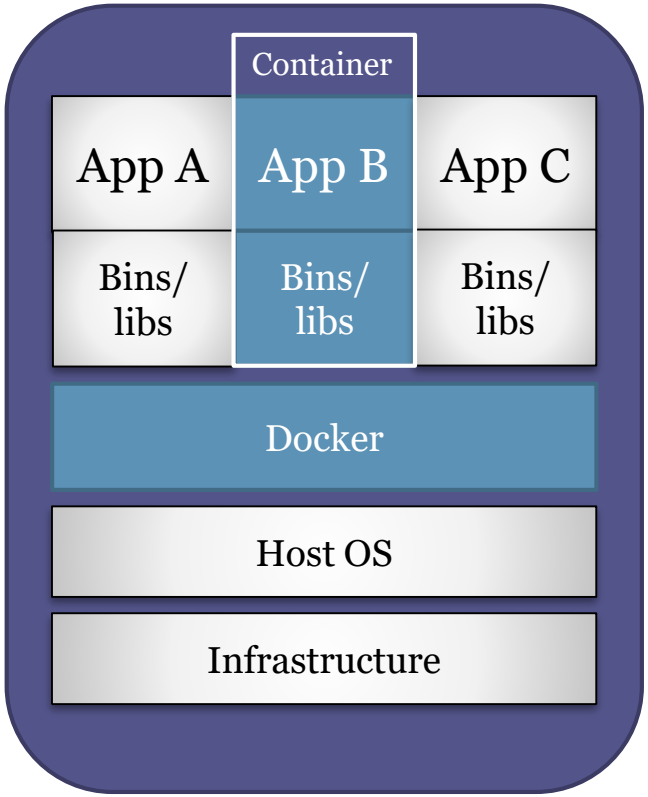


Execution environments

- Physical Hosts
  - Big computer vs server farms
- Virtual machines
  - Multiple OS can exist in the same machine
  - Provide portability and isolation
  - Very popular
    - Most applications run on virtual machines
  - Performance can be less predictable
- Containers
  - Docker
  - Local execution of processes
  - Images distributed in containers

Virtual machines vs Containers





Containers

- Advantages
  - Performance: less overhead and system resources
  - Increased portability
  - Easy installation: Deployment as code
  - Adapt very well to microservices
- Challenges:
  - Management:
    - Containerized applications can have lots of instances
    - Coordination between containers
    - Approaches: Kubernetes, Docker swarm

Continuous delivery

Frequent releases to obtain feedback as soon as possible

- Deployment pipeline

Advantages:

- Embrace change
- Minimize integration risks

Wabi-sabi philosophy
Accept imperfection
Software that is not finnished: Good enough

Continuous deployment

Deployment pipeline: Automated implementation of an application's build, deploy, test and release process

Goals

- *Create runtime environments on demand*
- *Fast, reliable, repeatable and predictable outcomes*
- *Consistent environments in staging and production*
- *Establish fast feedback loops to react upon*
- *Make release days riskless, almost boring*

Deployment pipeline

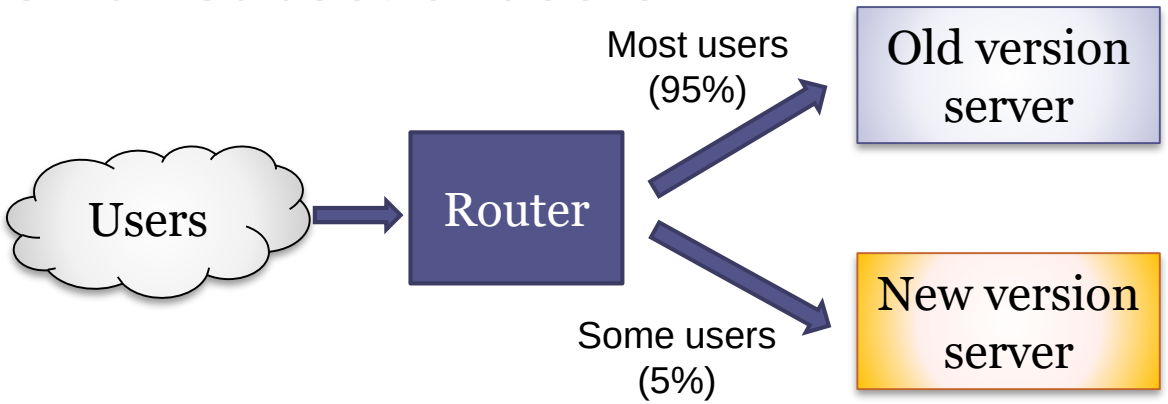
- Patterns
  - Infrastructure as code
  - Keep everything in Version Control
    - Code
    - Configuration
    - Data
  - Align development and operations (DevOps)
- Tools:
  - Ansible, Chef, Puppet,…
- Best practices: 12 factors (next slide)

12 factor https://12factor.net/

I. Codebase One codebase tracked in revision control, many deploys
II. Dependencies Explicitly declare and isolate dependencies
III. Config Store config in the environment
IV. Backing services Treat backing services as attached resources
V. Build, release, run Strictly separate build and run stages
VI. Processes Execute the app as one or more stateless processes
VII. Port binding Export services via port binding
VIII. Concurrency Scale out via the process model
IX. Disposability Maximize robustness with fast startup and graceful shutdown
X. Dev/prod parity Keep development, staging, and production as similar as possible
XI. Logs Treat logs as event streams
XII. Admin processes Run admin/management tasks as one-off processes

Testing and continuous delivery

- Feature toggles
  - Also known as *feature flags*, *feature bits*,...
  - Modify system behaviour without changing code
- Canary releases
  - Introduce new versions by slowly rolling out the change to small subset of users
- A/B testing



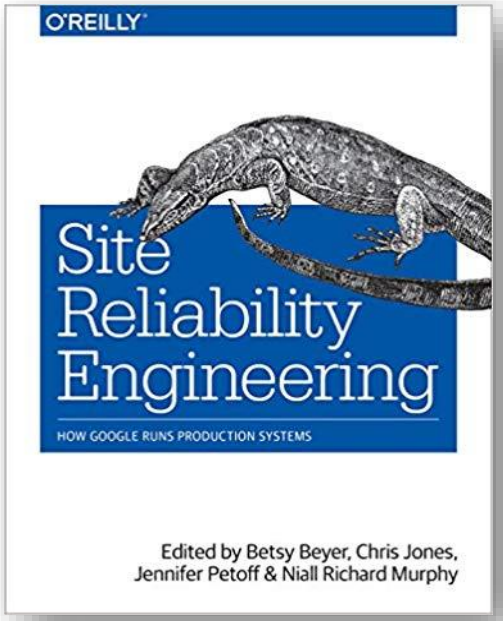
https://martinfowler.com/articles/feature-toggles.html
https://martinfowler.com/bliki/CanaryRelease.html

Software in production

Key quality attributes:

- Availability
- Reliability
- Observability

Recommended books



Free online

Reliability

- Capacity planning
  - Load testing
  - Example: JMeter, Gatling
- Load balancing
  - Increase reliability through redundancy
  - Failover

Logging and monitoring

- Quality attribute: Observability
  - Usually not required by customer
- Logging
  - Usually easy to generate
  - Logging as stream processing: Apache Kafka
- Metrics & Monitoring
  - Time-series database systems & visualizations
    - Prometheus, Graphite, Grafana, Datadog, Nagios, ...
  - Health checks

Incidents & post-mortem

- Resolve and review incident
- Ensure team view it as **blameless**
- Create post-mortem report
  - Incident details
  - Root Cause Analysis
  - Timeline and actions taken to resolve it
- Identify preventive measures

https://landing.google.com/sre/sre-book/chapters/postmortem-culture/

Chaos engineering

- Started by Netflix in 2010 (Chaos Monkey)
- Test distributed systems
  - Break things on purpose
    - Failure injection testing
    - Ensure that one instance failure doesn't affect the system
  - Antifragility and resilience

https://github.com/Netflix/chaosmonkey

End of presentation