From Legacy to Kubernetes: The SOA Transformation

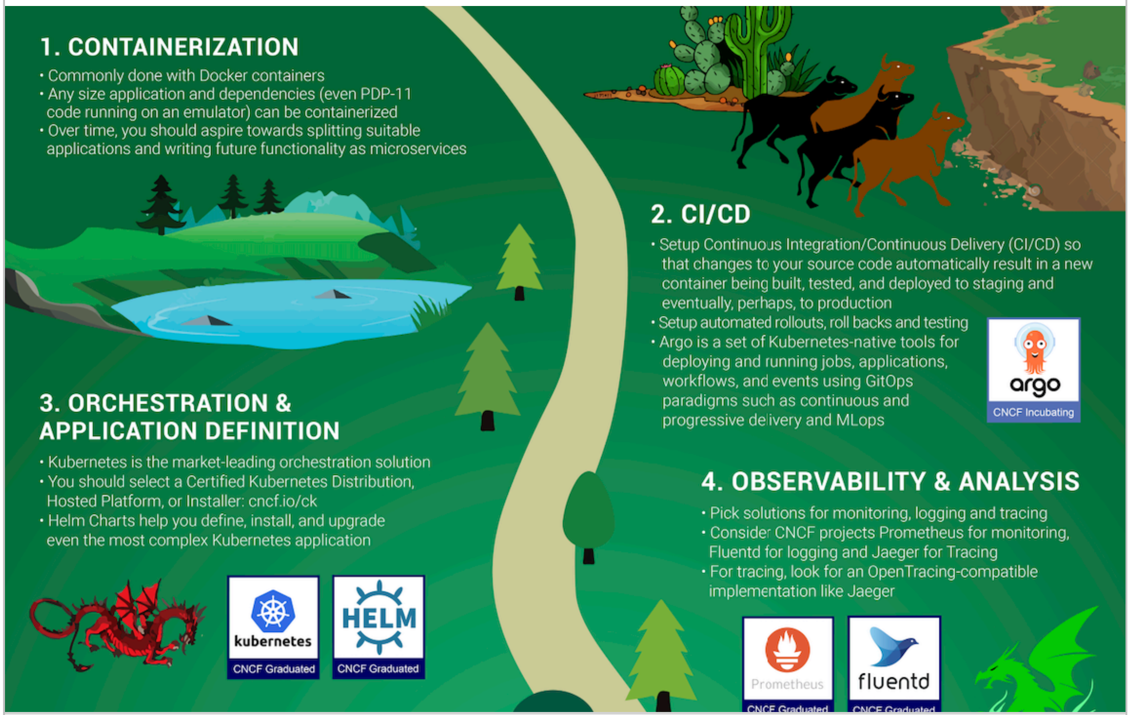
Assoc. Prof. Tomas Vitvar, Ph.D.
tomas.vitvar@fit.cvut.cz • https://vitvar.com

Czech Technical University in Prague
Faculty of Information Technologies • Web Intelligence Research Group • https://vitvar.com

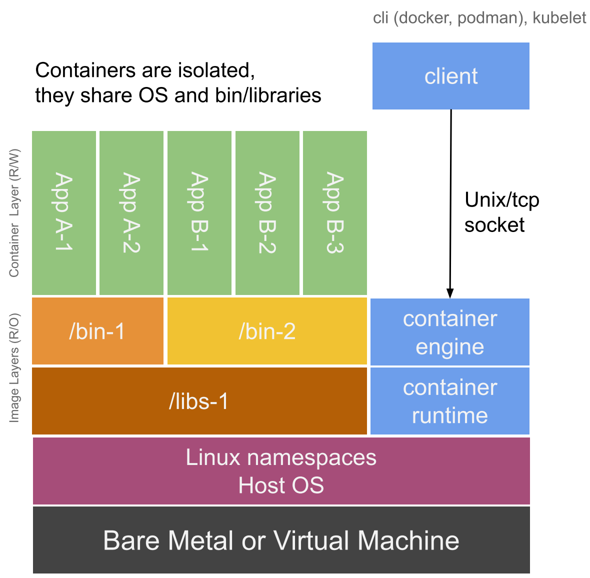
Overview

- Cloud Native
  - *Containers*
  - *Kubernetes*
- Service Oriented Architecture
- SOA Transformation
- Conclusion

CNCF Trail Map



Container Stack



Linux Namespaces

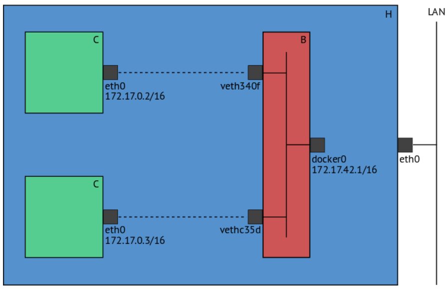
- Isolation of Linux processes, there are **8 namespaces** (kernel v5.6)
  - *Mount, UTS, IPC, PID, Network, User, Cgroup, time*
  - *By default, every process is a member of a default namespace of each type*
  - *A container runtime starts a proces in separate namespaces of each type*
  - *The process children are started in the same namespaces*
  - *Run* lsns *to check namespaces the process is in*

```
$ lsns
NS         TYPE  NPROCS   PID USER   COMMAND
4026531836 pid        2 30873 oracle -bash
4026531837 user     108  1636 oracle /bin/bash /u01/oracle/scripts/startWebLogicContainer
4026531838 uts        2 30873 oracle -bash
4026531839 ipc        2 30873 oracle -bash
4026531840 mnt        2 30873 oracle -bash
4026531956 net      108  1636 oracle /bin/bash /u01/oracle/scripts/startWebLogicContainer
4026532185 mnt       13 13542 oracle /bin/bash /u01/oracle/scripts/startNM_ohs.sh
4026532192 pid       13  2798 oracle /bin/bash /u01/oracle/scripts/startNM_ohs.sh
...
```

- Flexible configuration, for example:
  - *You can run two apps that only share the network namespace, e.g.* 4026531956
  - *The apps can talk to each other*
  - *Any other app (not in this namespace) won't be able to talk to the apps*

Types: net

- net namespace
  - *Processes have their own private network stack (interfaces, routing tables, sockets)*
  - *Communication with external network stack is done by a virtual ethernet bridge*

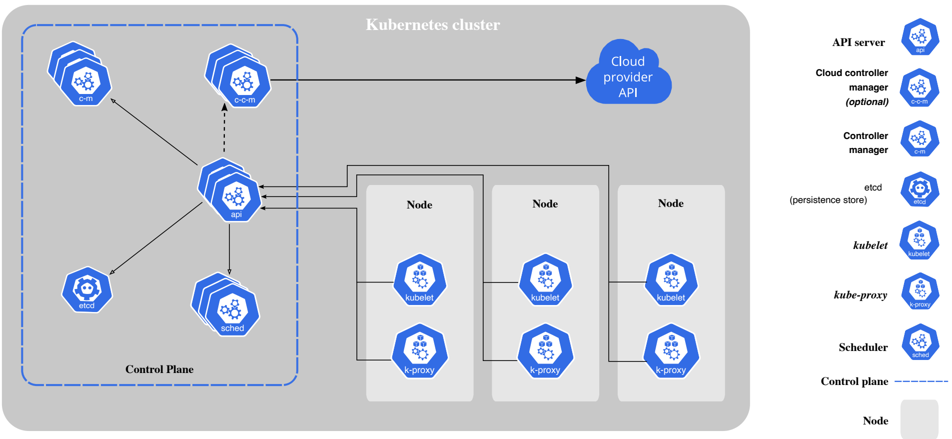


  - *On the host there is a* ***userland proxy*** *or* ***NAT***
    - → *NAT is a prefered solution over userland proxy (*/usr/bin/docker-proxy*)*
    - → *Lack of NAT hairpinning may prevent to use NAT*
  - *Use case*
    - → *Multiple services binding to the same port on a single machine, e.g.* tcp/80
    - → *A port in the host is mapped to the port exposed by a process in the NS*

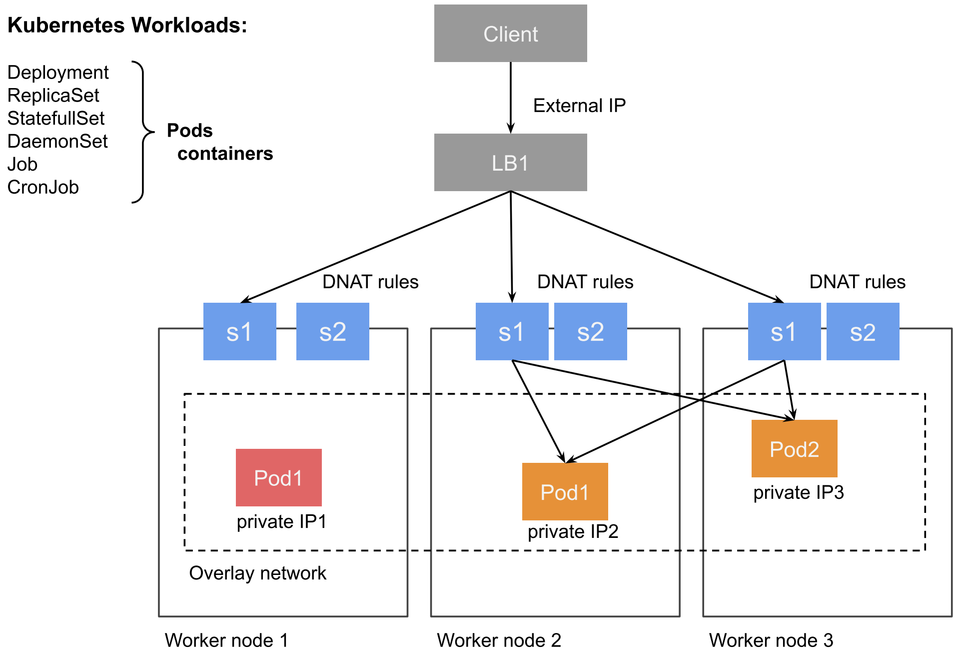
Overview

- Cloud Native
  - *Containers*
  - *Kubernetes*
- Service Oriented Architecture
- SOA Transformation
- Conclusion

Kubernetes Architecture



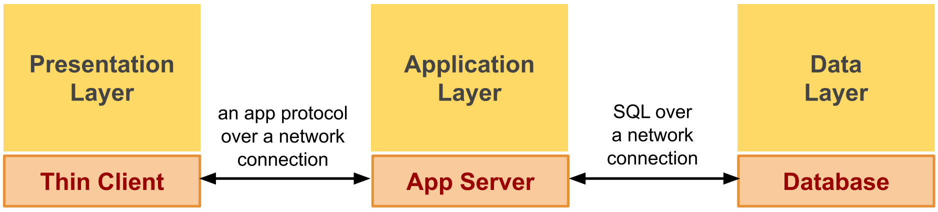
Overlay Network



Overview

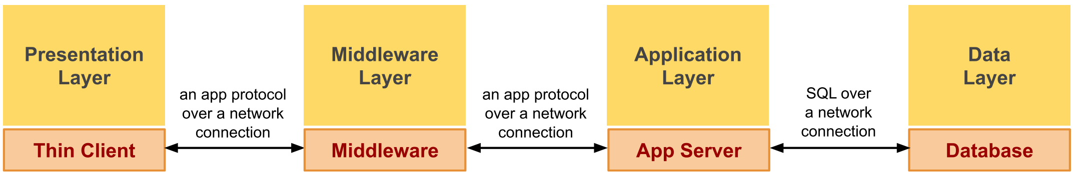
- Cloud Native
- Service Oriented Architecture
  - *Architectures evolution*
  - *Enterprise Service Bus*
- SOA Transformation
- Conclusion

Client/Server Architecture (three-tier)



- All layers on separated machines
  - *Thin client – browser*
  - *Monolithic architectures*
- Drawbacks
  - *"spaghetti" integration*
  - *Limited, single app server scalability*

Client/Server Architecture (multi-tier)

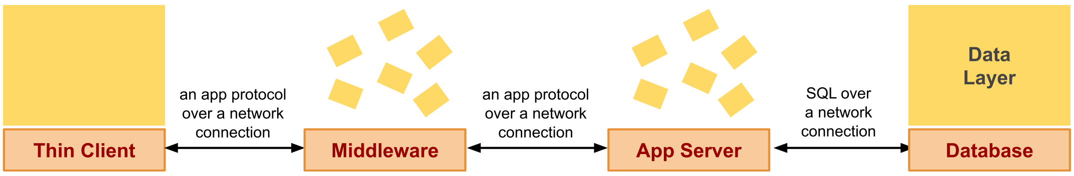


- A middleware layer
  - *Value-added services for communications*
  - *Load balancer, proxy server, auth* servers, ..., integration middleware*
- Drawbacks
  - *Still monolithic architectures*
  - *Deployment overhead, scalability and performance issues*

Client/Server Architecture (microservices)

- Microservice architecture
  - *Middleware, app and DB monoliths are microservice architecture*
  - *Improved scalability and technology neutrality of app components*
- Service orchestration layer
  - *Kubernetes (K8s)*
  - *Large K8s cluster for all, middleware, app, DB*
  - *Separate K8s cluster*

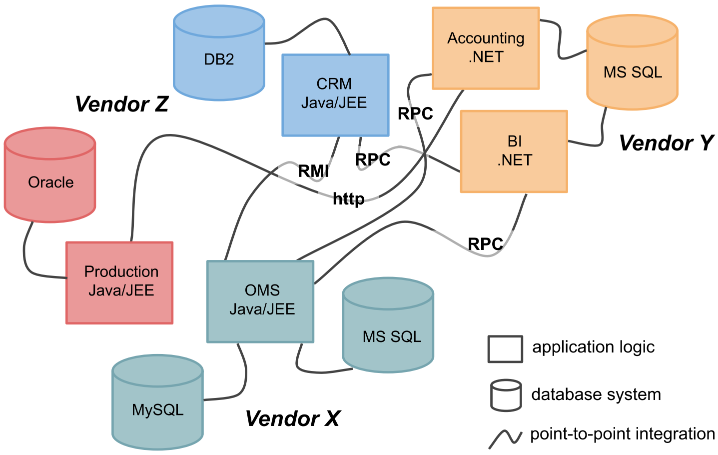
Client/Server Architecture (~~microservices~~)



- Not-a-microservice Architecture
  - *Monoliths deployed to Kubernetes cluster*
  - *Improved Deployments (via container images)*
  - *Improved fail-over*
  - *Not cheaper (Kubernetes costs come into play)*

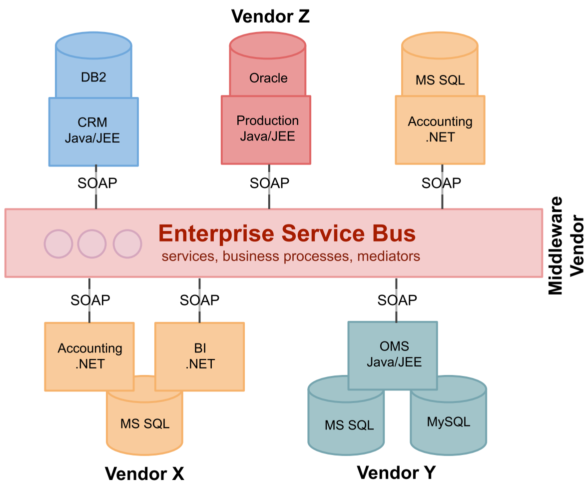
Spaghetti Integration

- Integration is a chaos
  - *Multiple protocols problem, multiple vendors*
  - *Ad-hoc integrations*



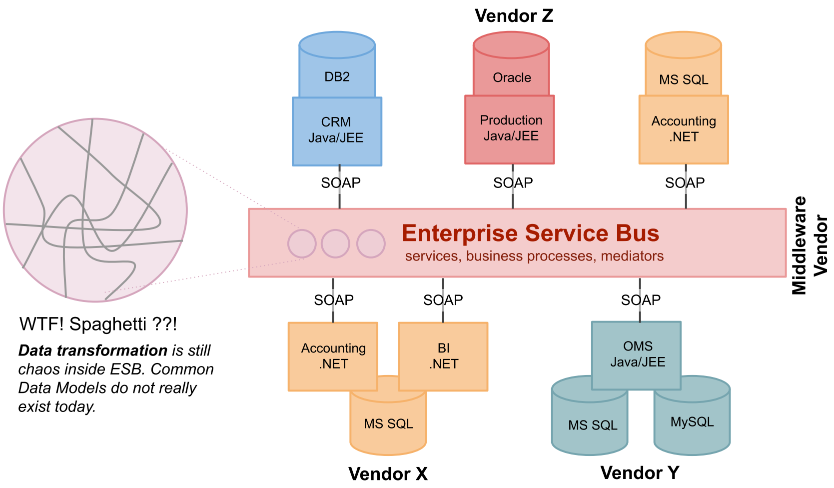
Service Oriented Architecture

- Integration is organized
  - *SOA Governance*
  - *Enterprise Service Bus in the core of the architecture*
    - → *Monolithic system, many components to realize integrations*



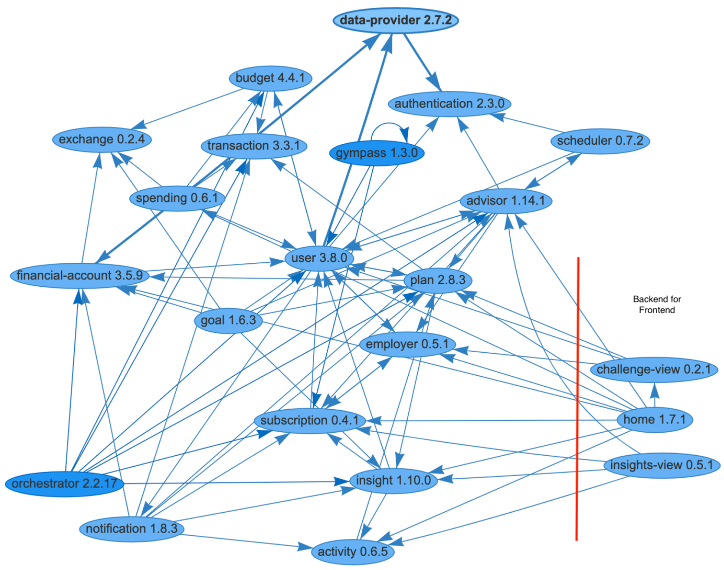
Service Oriented Architecture

- Integration is organized
  - *SOA Governance*
  - *Enterprise Service Bus in the core of the architecture*
    → *Monolithic system, many components to realize integrations*



Back to the Stone Age

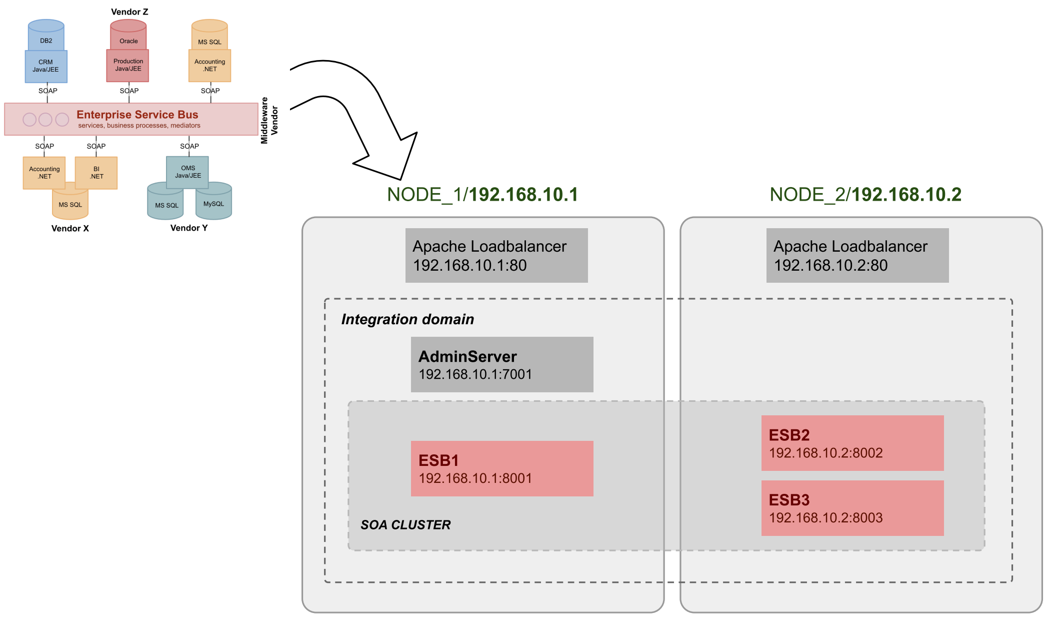
- A microservice architecture today
  - *WTF! Spaghetti??!*



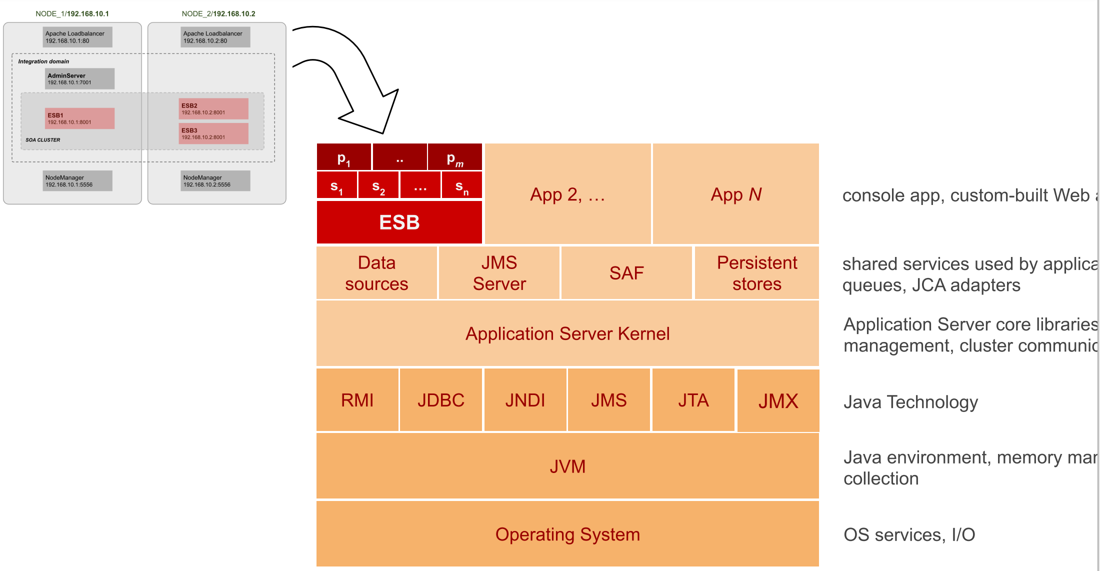
Overview

- Cloud Native
- Service Oriented Architecture
  – *Architectures evolution*
  – *Enterprise Service Bus*
- SOA Transformation
- Conclusion

ESB Topology

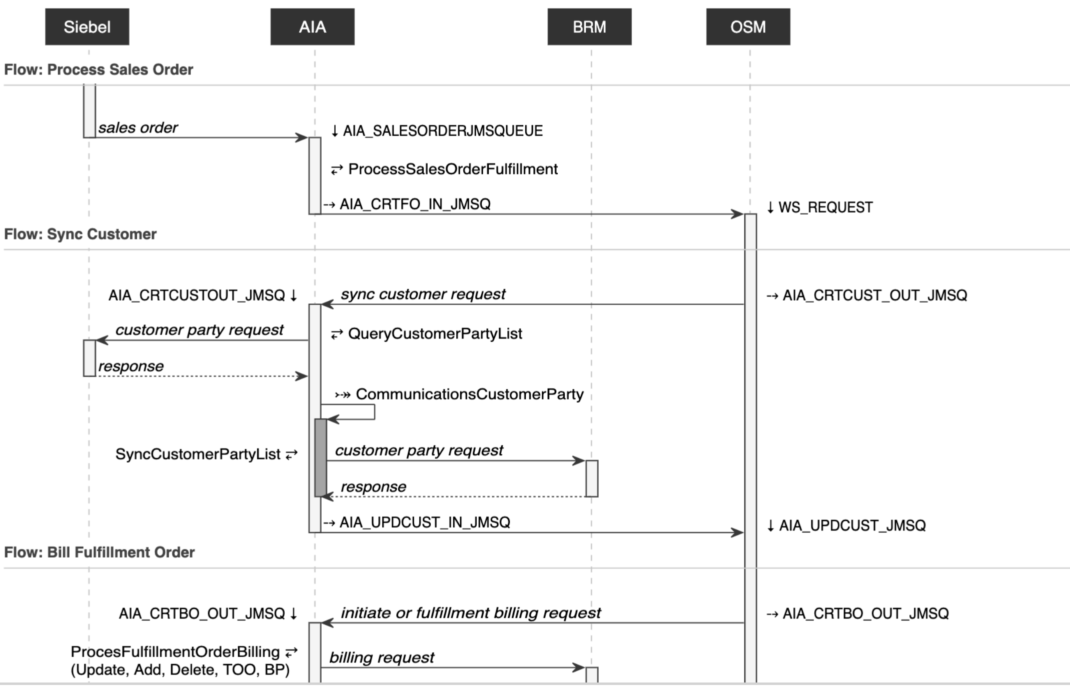


Application Server Layers with ESB



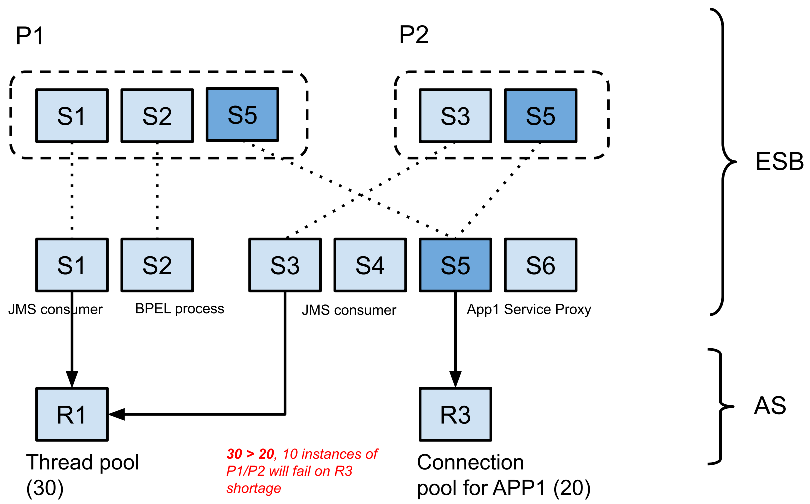
- ESB is instance running on a node
- JVM process

Order to Cash Integration Process Example



Resources Constraints

- Connection Pools, Thread Pools
  - *ESB has a pool of connections to a back-end app (R3)*
  - *A single connection is resued by multiple instances of processes*

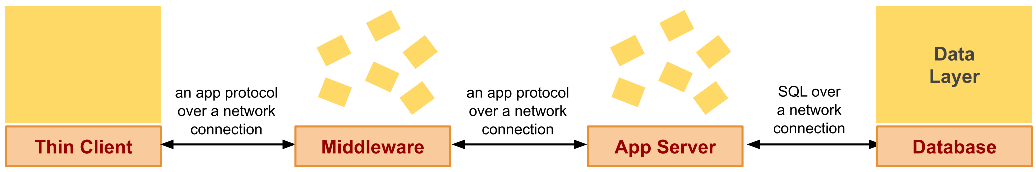


Overview

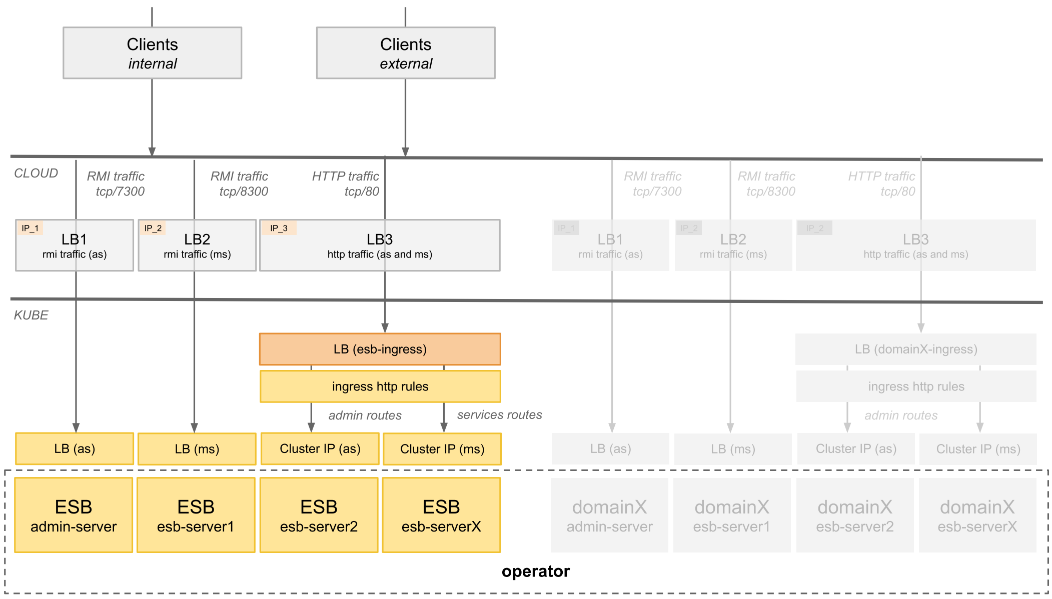
- Cloud Native
- Service Oriented Architecture
- SOA Transformation
- Conclusion

Overview

- BSS Stack
  - *Applications that support business operations*
  - *CRM, BRM, OMS, IDAM, Inventory, ...*
- Large deployemnts
  - *Very large customer base*
  - *~500 services*
  - *~100 requests/second per service*
- Target architecture
  - *Kubernetes to run apps in BSS*
  - *No microservices, monolithic architecture preserved*



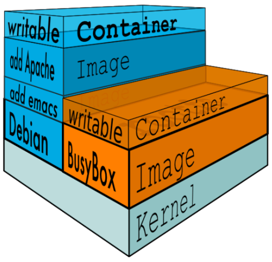
Architecture



Wrong assumptions

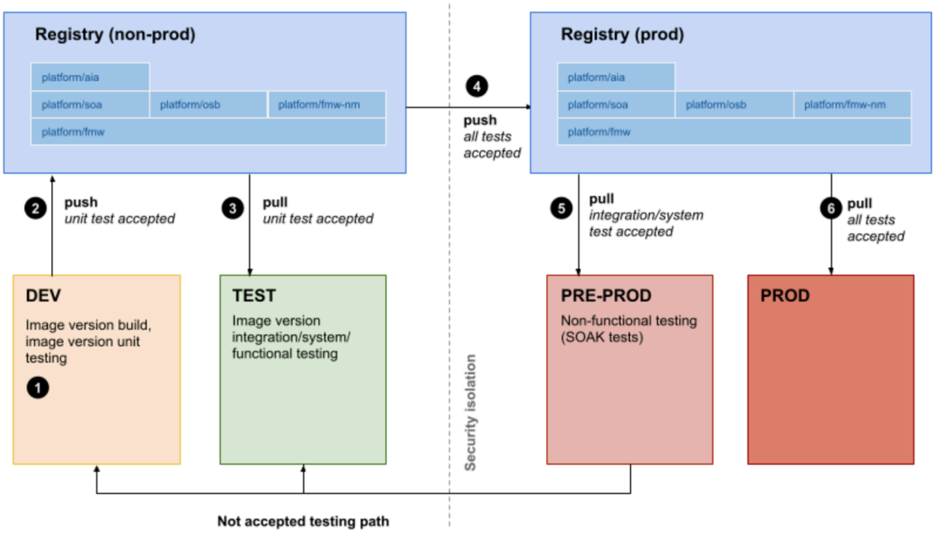
- SOA becomes a microservice architecture when it is running in Kubernetes
  - *No, SOA will still be a monolithic architecture even with Kubernetes*
- It will be possible to auto-scale SOA in Kubernetes
  - *No, auto-scaling of SOA depends on auto-scaling capabilities of SOA, not Kubernetes*
- SOA will be more reliable and resilient with Kubernetes
  - *SOA is reliabile and resilient as SOA is, not Kubernetes*

Storage and Persistence

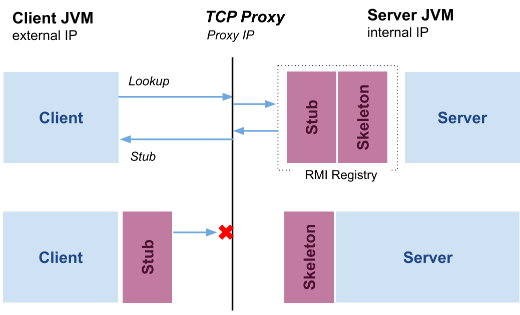


- Image layers and container layer
  - *Images are immutable*
  - *Application server and ESB binaries*
- Persistent Volumes
  - *Mounted to containers*
  - *Operational data (logs, transaction logs), ESB configurations*
- Database
  - *Application services*

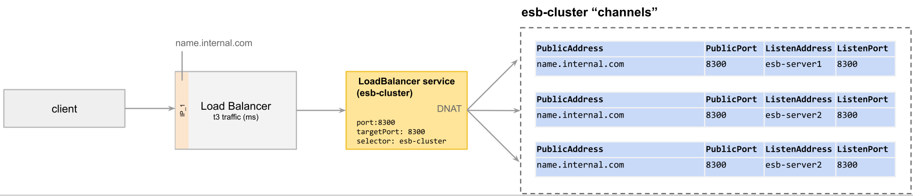
Container image lifecycle



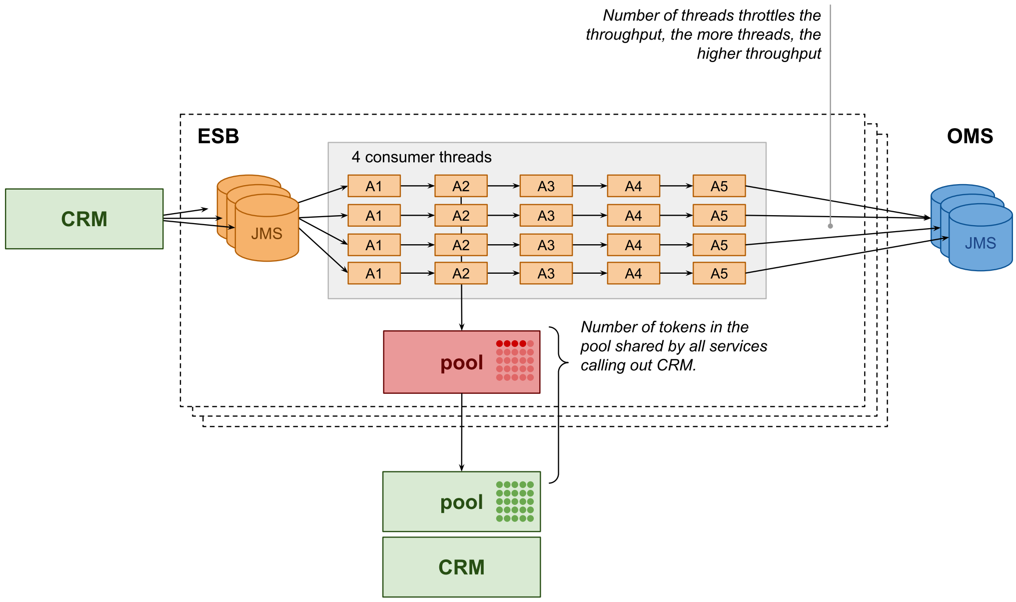
RMI Networking



- RMI requires a special setup
  - *Any external JMS call goes via RMI*
  - *Extensive usage of RMI from external clients*
  - *Requires additional "channel" configuration in ESB*



Auto-scaling, not really



Overview

- Cloud Native
- Service Oriented Architecture
- SOA Transformation
- Conclusion

Summary

- Phased approach
  - *Containerization first*
  - *CI/CD to build, test, deploy*
  - *Kuberneres next*
- Advantages
  - *A step towards Kubernetes, ready to build real microservices*
  - *Teams' awarness, they learn to develop, test and operate new platform*
  - *Improved deployments (container images)*
- Challenges
  - *Networking, technology-specific protocols*
  - *Deployments (filesystem, DB)*
  - *How to run old style apps (desktop)*
  - *Preserve ESB topology*