

Lecture 14 - recap

Kari Systä, 01.12.2020

About the exam

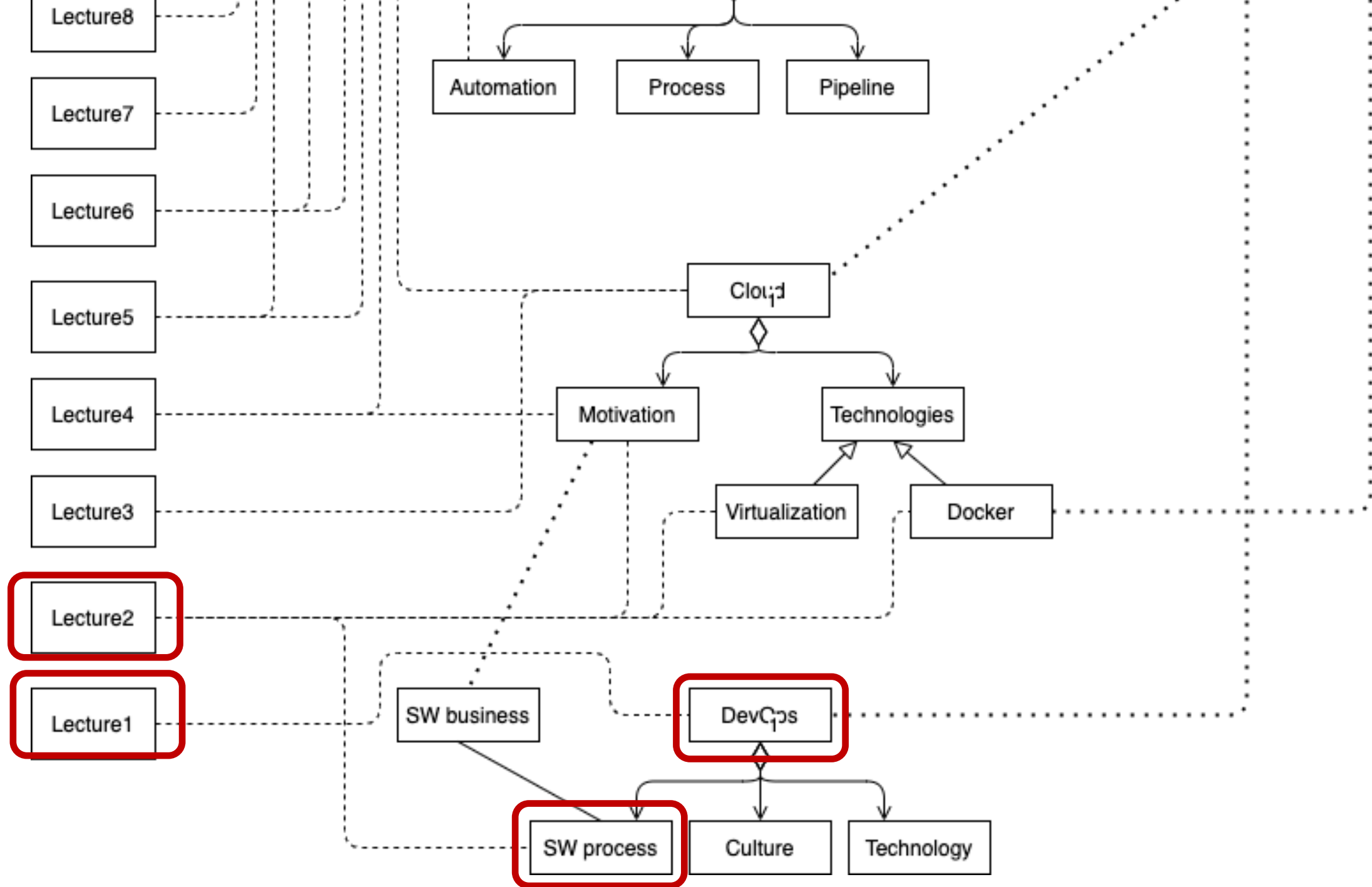
- In SISU I have 24 confirmed registrations
- Then, I see 52 “planned” registrations. These are assumed not to participate.
- Please check your status
 - I hope you know how – I don’t. Should be in your “calendar”-view
- Also a security notes:
 - “a student who according to his or her own estimate is (or who shares accommodation with someone who is) at high risk for severe illness from COVID-19 will not be required to attend classes or other activities (e.g. exam) on campus, but we arrange an alternative mode of completion.”
 - The “COVID-19 committee” will have a meeting today – so changes in the plans are possible

About project

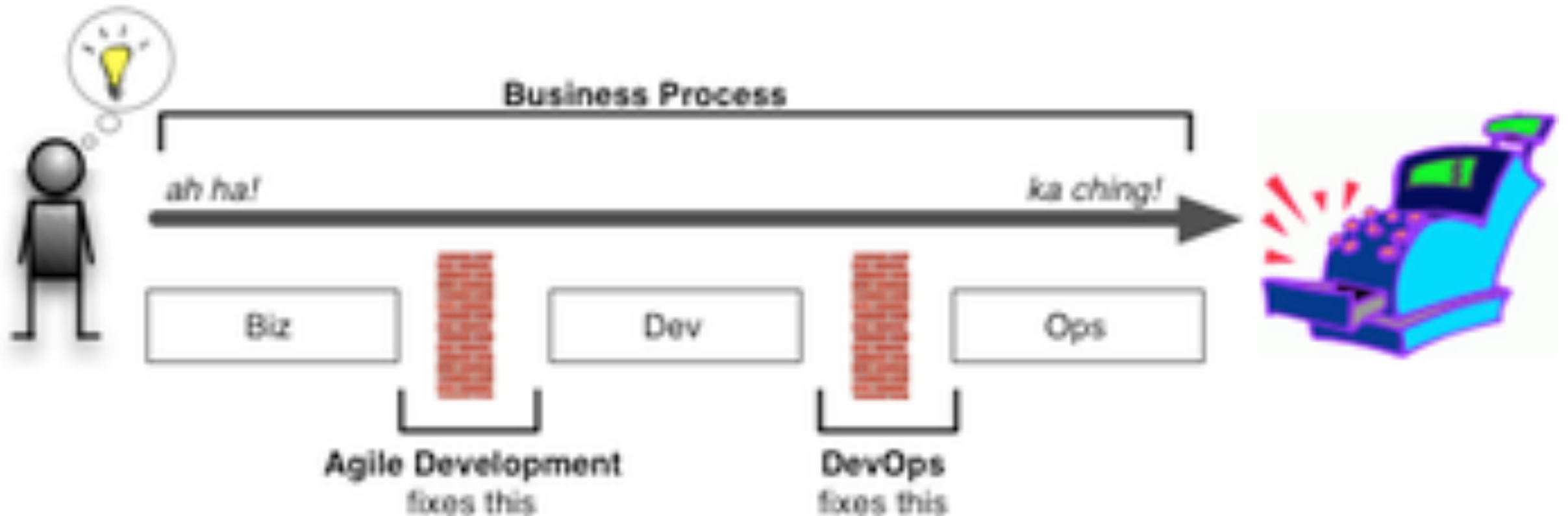
- Some of you have experienced various problems in running docker-compose in gitlab-runner. This problem did not surface last year.
- Unfortunately, the course staff have not had time to dive in to this problem, but we hope
 - You visit Kooritorio so that aspects of the problem are well understood.
 - Find time to investigate the problem next week.
- There are also other approaches, like
 - A separate daemon in host that runs the docker-compose
 - A separate VM to do it
 - ...

A job

- Atlas Electric is looking for a master thesis worker.
- Topic: selection and integration of a map component:
teemu.nieminen@fi.atlas-elektronik.com



The lifecycle



What is DevOps

(there are several definitions)

- Lucy Lwakatare:
 - DevOps is a concept that embodies a **cultural and mindset change** that is substantiated with a **set of practices** to encourage **cross-disciplinary collaboration between software development and IT operations** within a software company. The main purpose for the collaboration is to enable the **fast release of quality software changes while simultaneously operating resilient systems**.
 - From a **socio-technical perspective**, DevOps practices are focused on the **automation practices** of software deployment and infrastructure management, specifically automation of configuration management and monitoring.

DevOps practices

- Organizational
 - increased scope of responsibilities for developers;
 - intensified cooperation between development and operations.
- Technical
 - automation,
 - monitoring
 - measurement

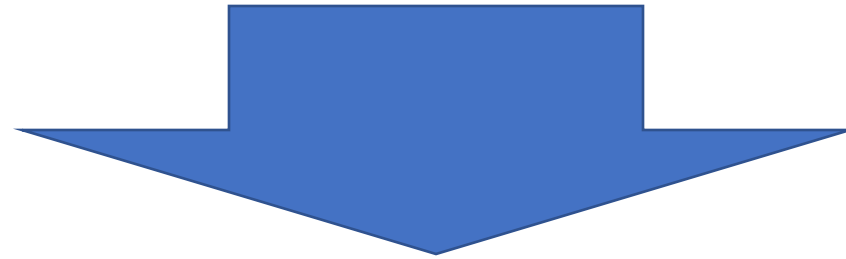
Where was the beef?

Business

Development

Operation

Use



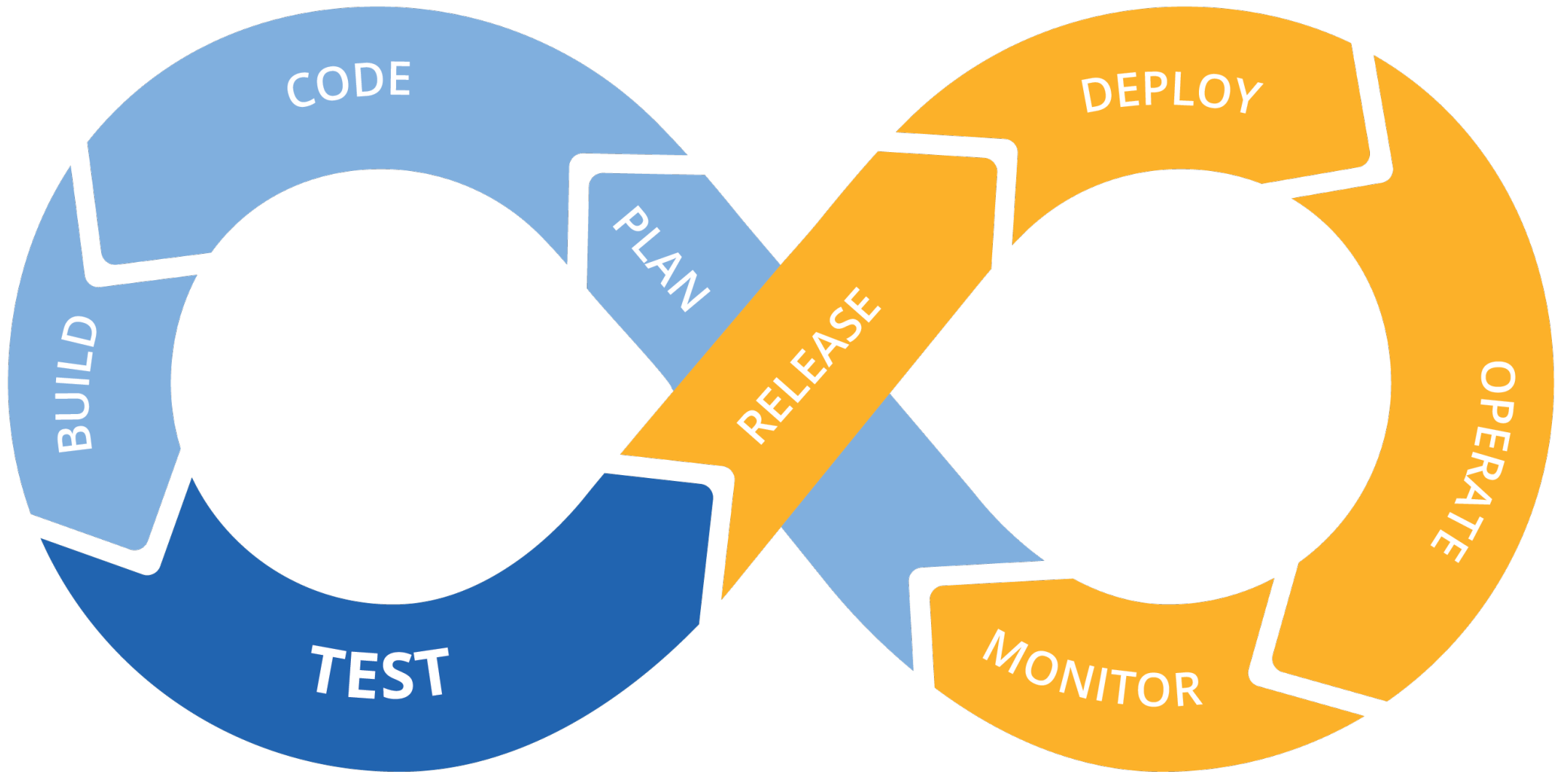
Business

Development

Operation

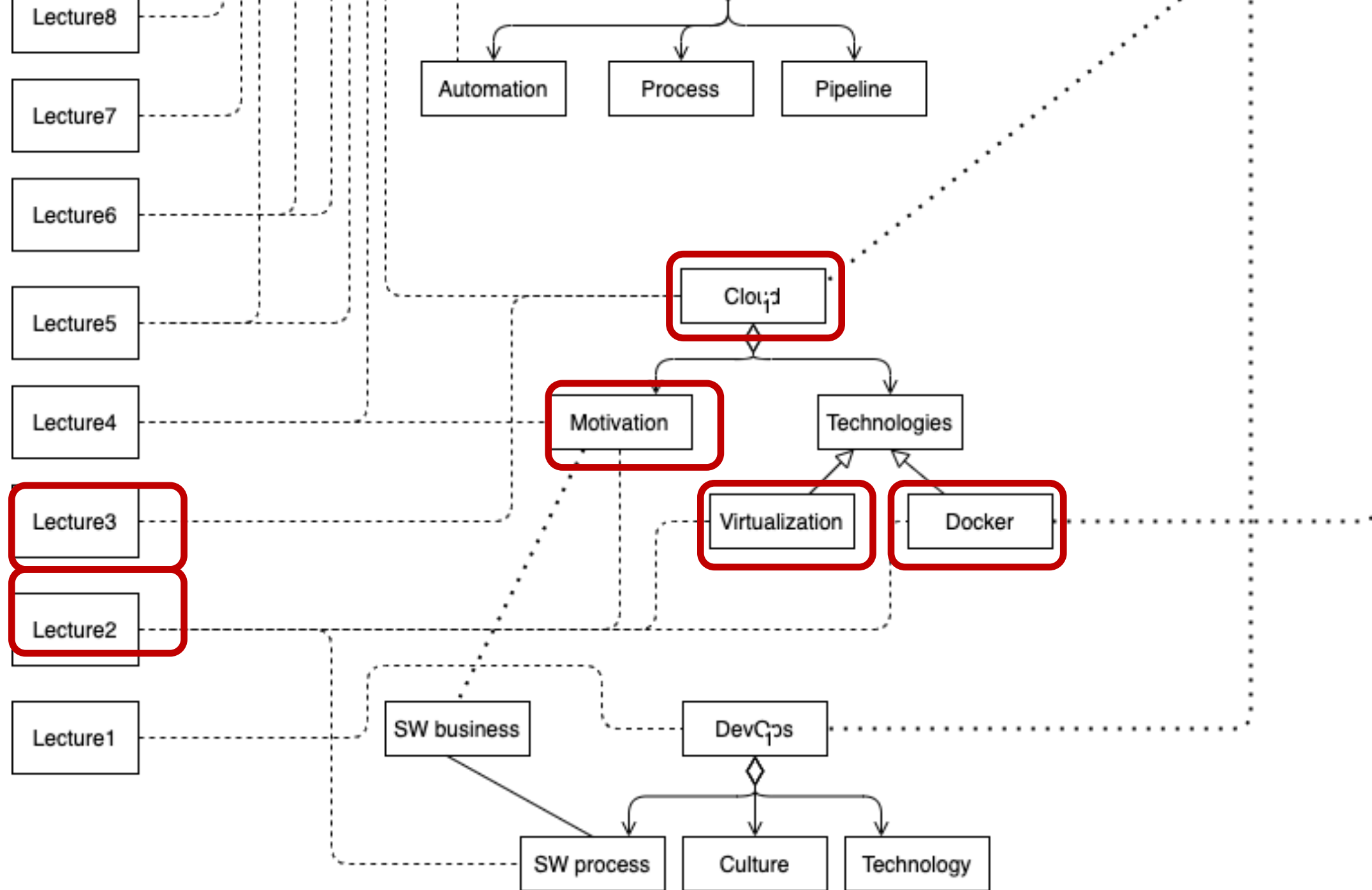
Use

DevOps



DevOps - benefits and challenges

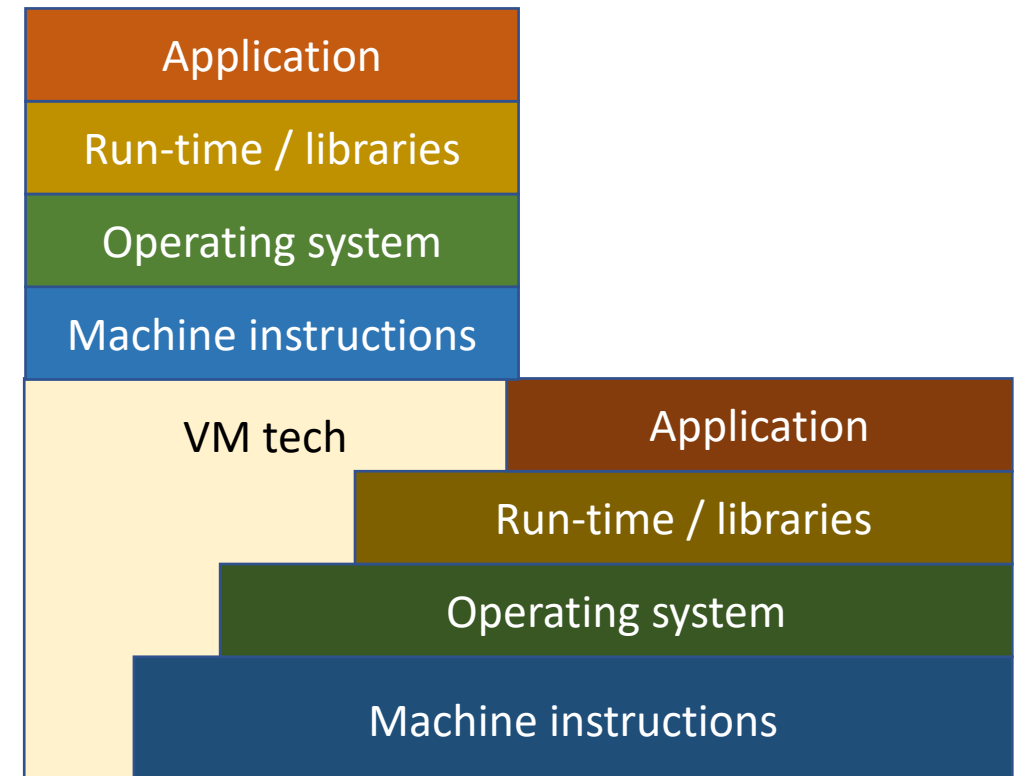
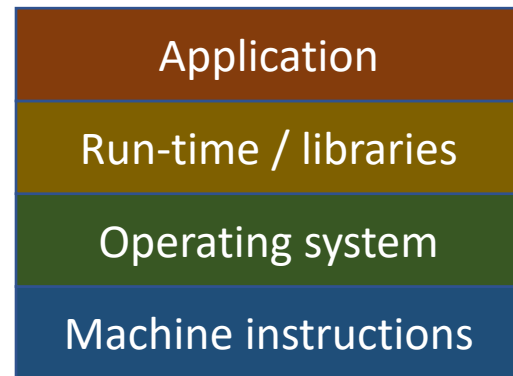
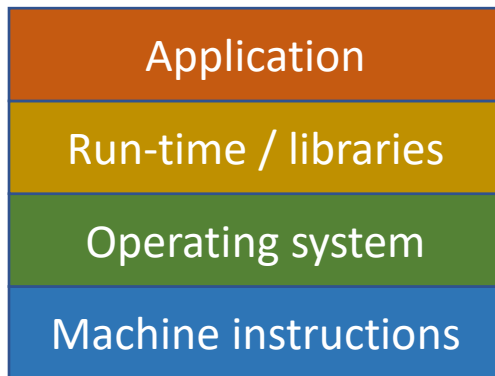
- **improvement in speed (release cycle time)**
 - **continuous deployment of system changes**
 - **productivity of operations work**
 - **improved morale, knowledge and skills**
-
- **resource constraints;**
 - **insufficiencies in infrastructure management;**
 - **high demands for required skills and knowledge, and**
 - **difficulties in monitoring microservices**



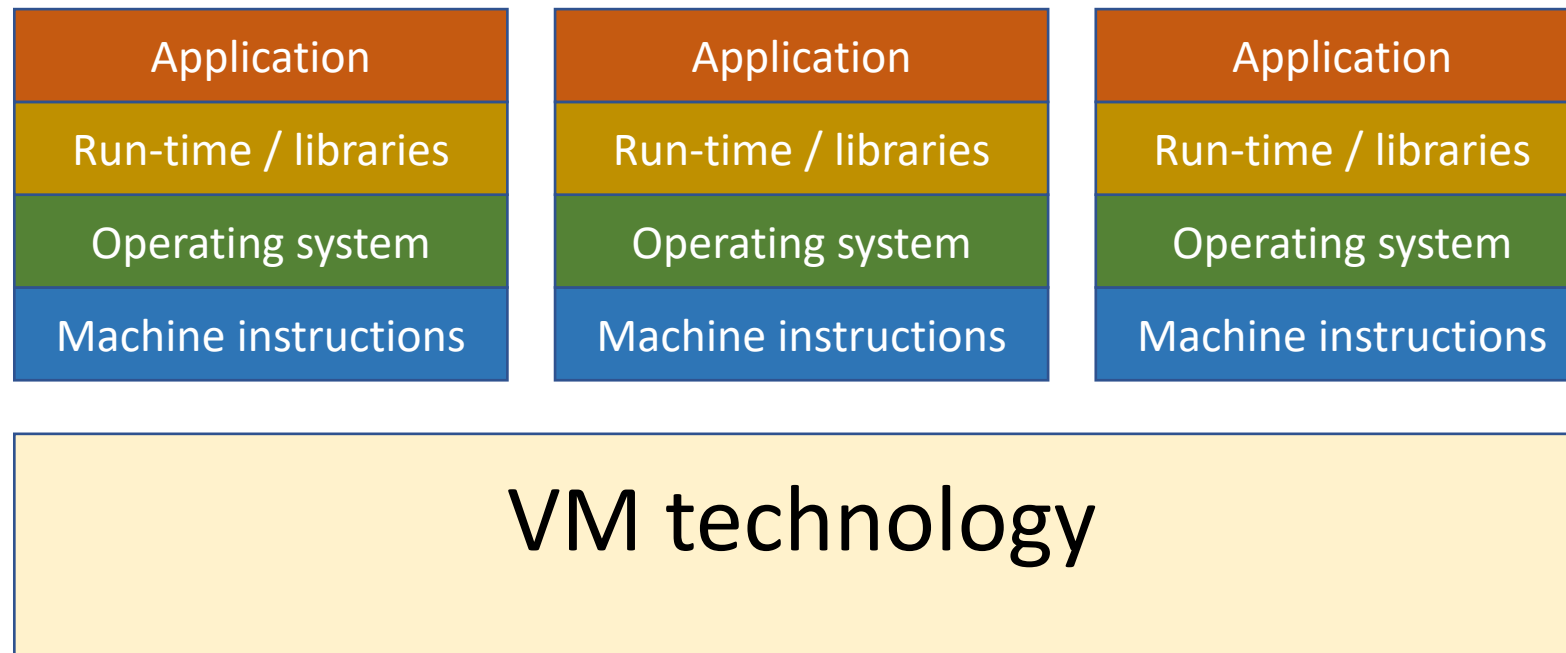
Levels of virtualization

- Hardware virtualization
- Operating system virtualization
- Desktop virtualization
- Application virtualization
- Network virtualization

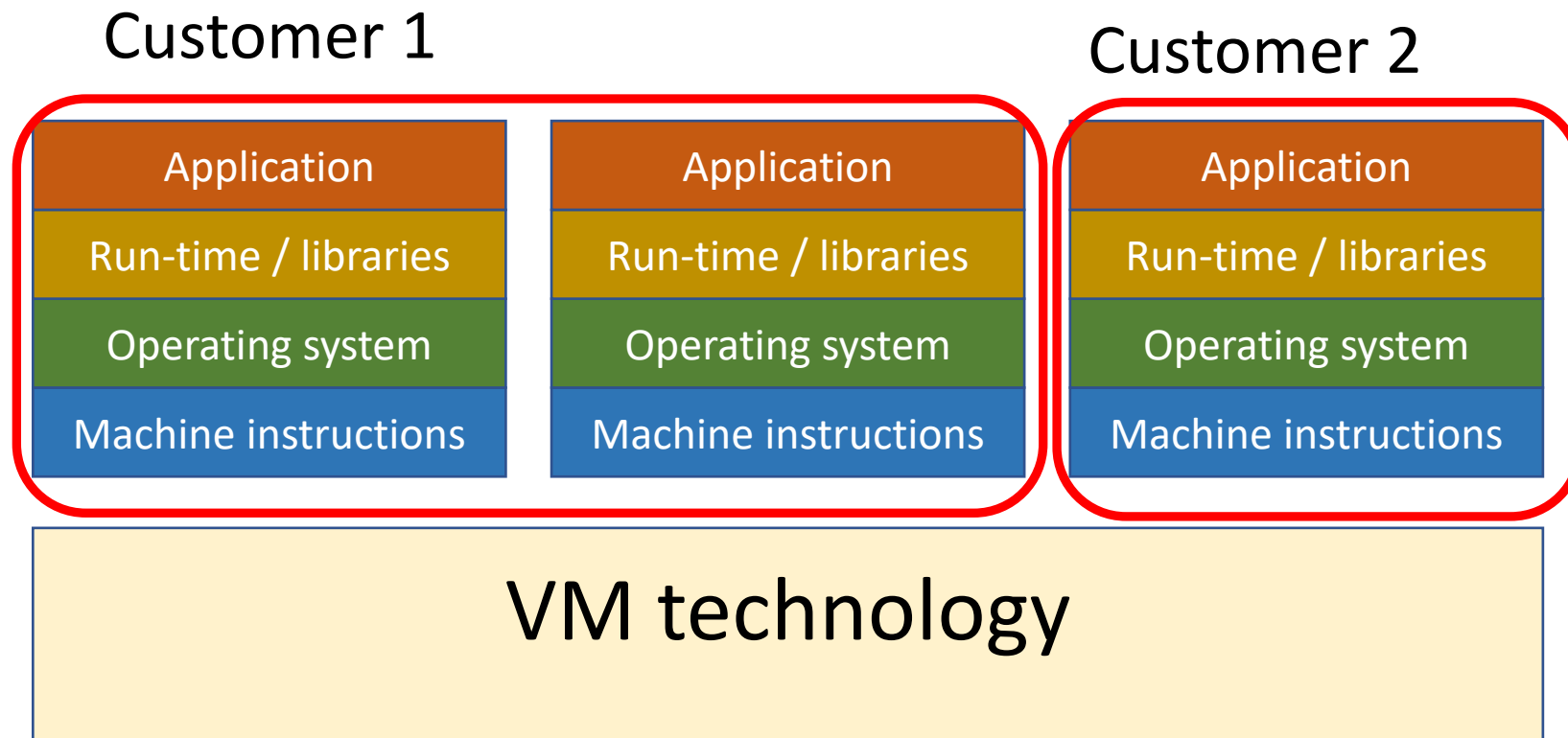
Use case 1: run "foreign" software



Use case 2: isolate



Use case 3: scale



Cloud computing - definition

- In 1997, Professor Ramnath Chellapa of Emory University defined Cloud Computing as the new *'computing paradigm, where the boundaries of computing will be determined by economic rationale, rather than technical limits alone.'*
- *NIST*: Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

-

Essential characteristics 1/2

- *On-demand self-service.* A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
 - Unilaterally?
 - Without human interaction?
- *Broad network access.* Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
 - What does the heterogeneous platforms mean in practice?
- *Rapid elasticity.* Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
 - What of scaling is not automatic?

Essential characteristics 2/2

- *Resource pooling.* The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.
 - Why is this essential?
- *Measured service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.
 - Why?

Service models



A diagram showing three stacked rectangular boxes representing service models. The top box is brown and labeled 'SaaS'. The middle box is dark green and labeled 'PaaS'. The bottom box is blue and labeled 'IaaS'.

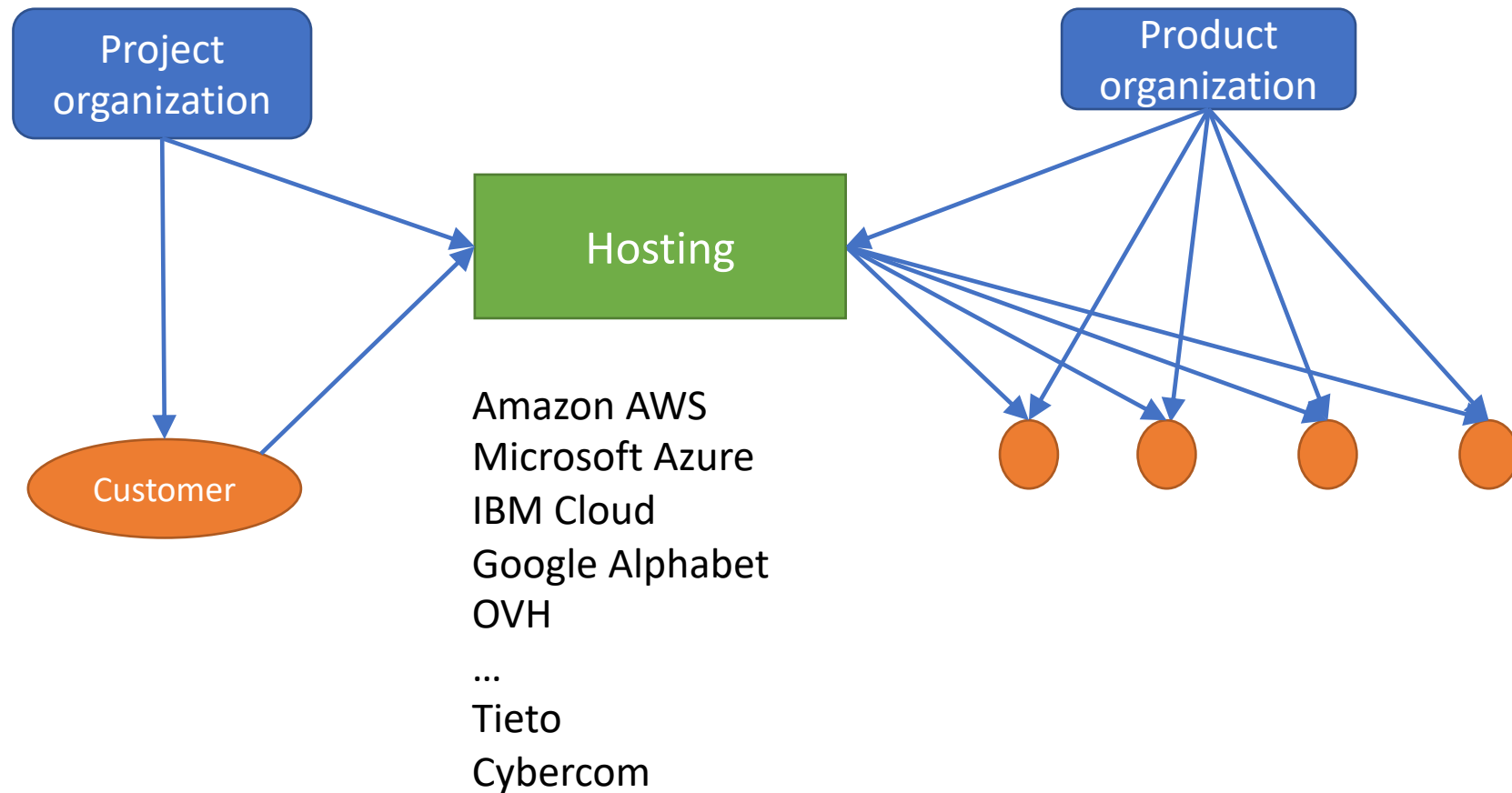
SaaS

PaaS

IaaS

Examples?

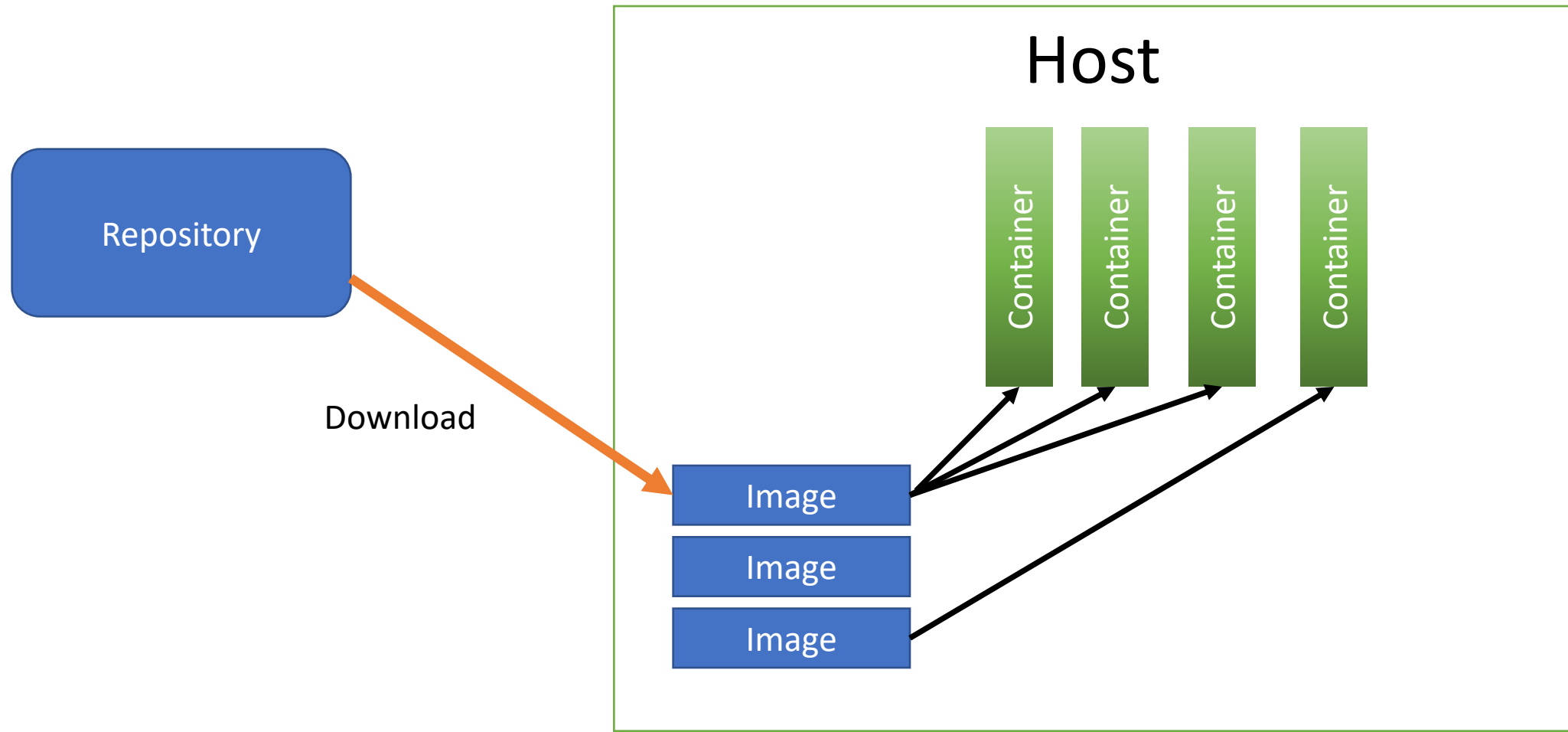
Hosting can be a separate business



<p>Pets are given names like grumpycat.petstore.com</p> <p>They are unique, lovingly hand raised and cared for</p> <p>When they get ill, you nurse them back to health</p>
<p>Infrastructure is a permanent fixture in the data center</p>
<p>Infrastructure takes days to create, are serviced weekly, maintained for years, and requires migration projects to move</p>
<p>Infrastructure is modified during maintenance hours and generally requires special privileges such as root access</p>
<p>Infrastructure requires several different teams to coordinate and provision the full environment</p>
<p>Infrastructure is static, requiring excess capacity to be dormant for use during peak periods of demands</p>
<p>Infrastructure is a capital expenditure that charges a fixed amount regardless of usage patterns</p>

<p>Cattle are given numbers like 10200713.cattlerancher.com</p> <p>They are almost identical to other cattle</p> <p>When they get ill, you replace them and get another</p>
<p>Infrastructure is stateless, ephemeral, and transient</p>
<p>Infrastructure is instantiated, modified, destroyed and recreated in minutes from scratch using automated scripts</p>
<p>Infrastructure uses version-controlled scripts to modify any service without requiring root access or privileged logins</p>
<p>Infrastructure is self-service with the ability to provision computing, network and storage services with a single click</p>
<p>Infrastructure is elastic and scales automatically, expanding and contracting on-demand to service peak usage periods</p>
<p>Infrastructure is an operating expenditure that charges only for services when they are consumed</p>

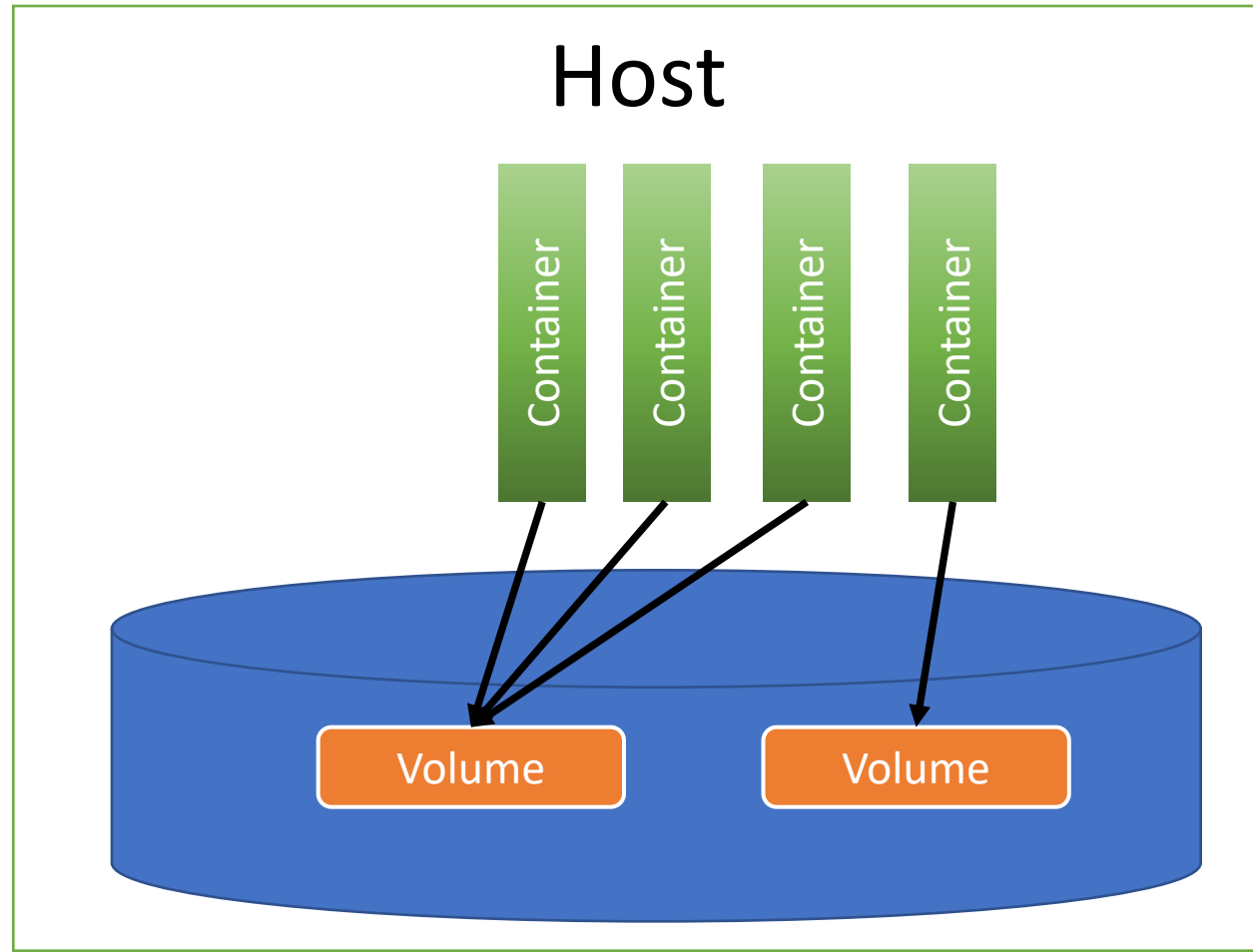
Typical set-up



Typical set-up

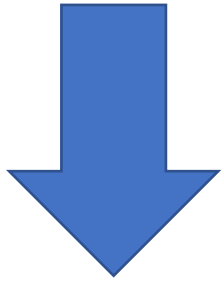
Volumes are for

- Persistent data
- Data sharing

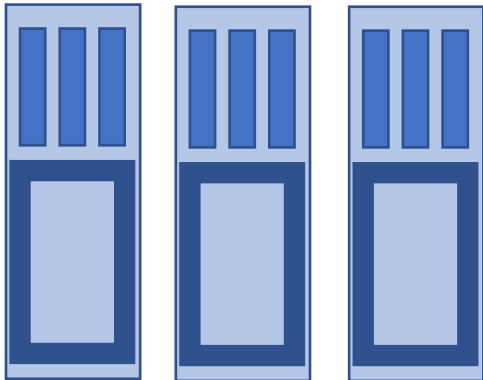


Docker swarm - docker compose

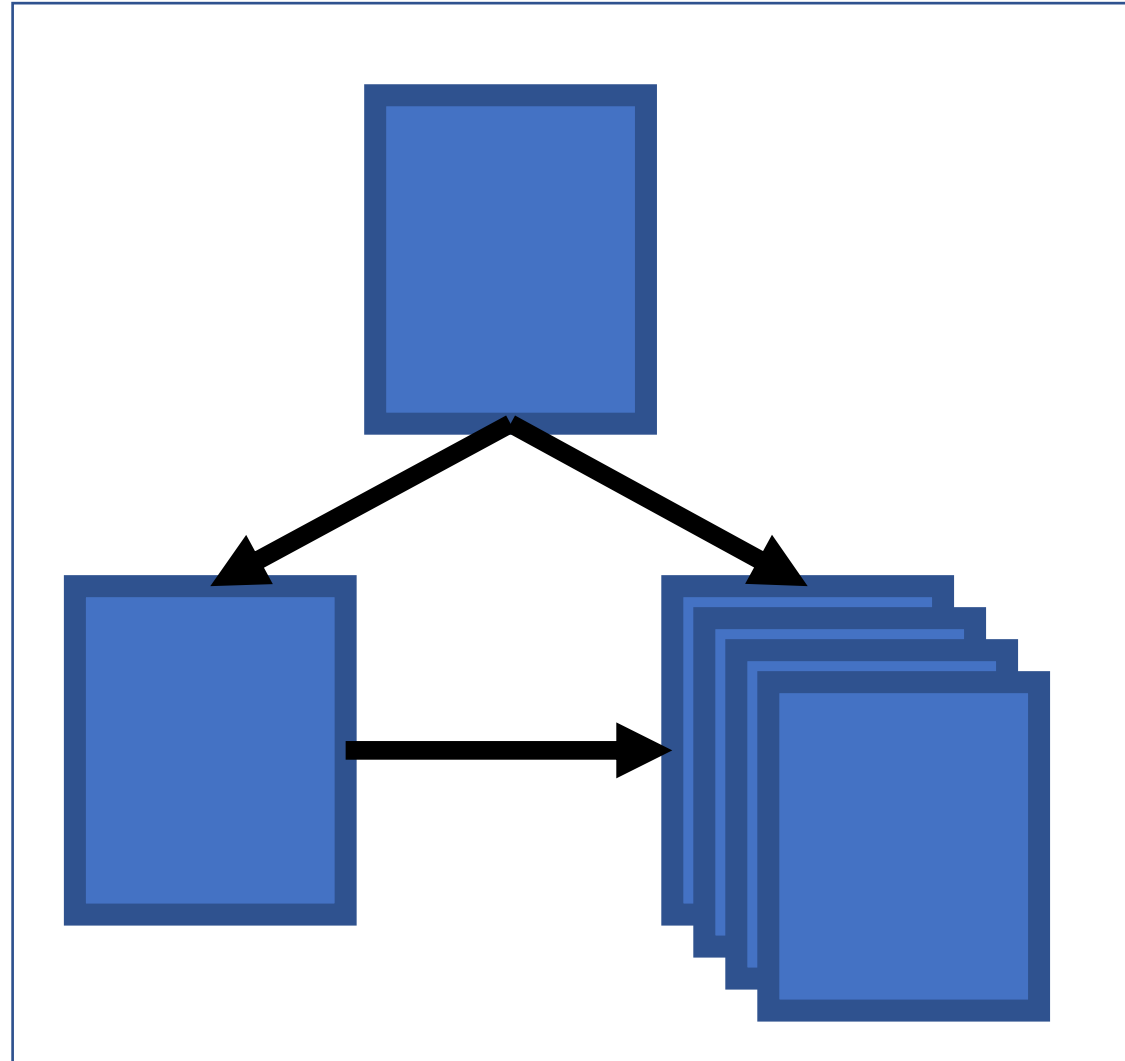
Orchestration



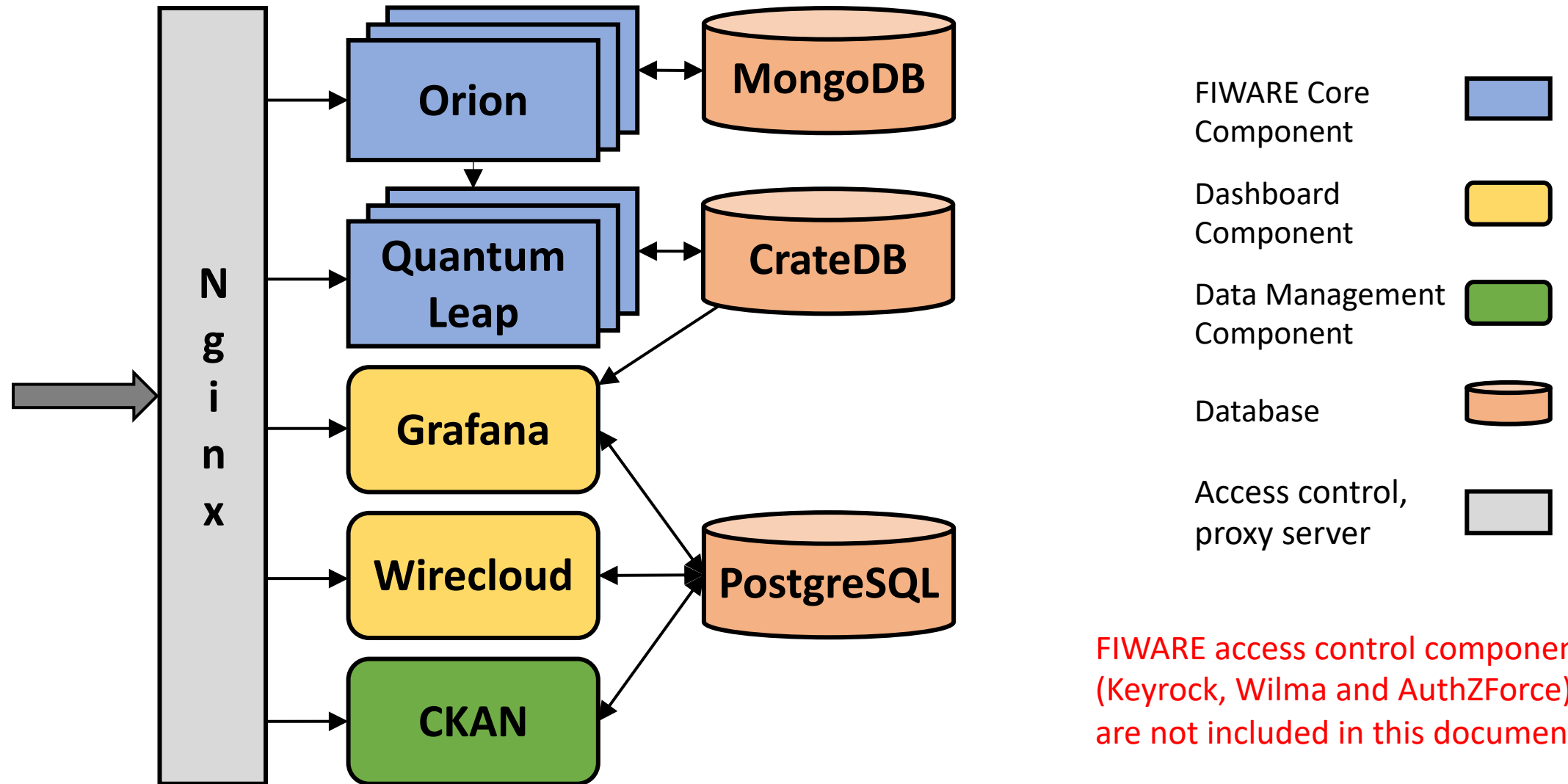
Docker swarm

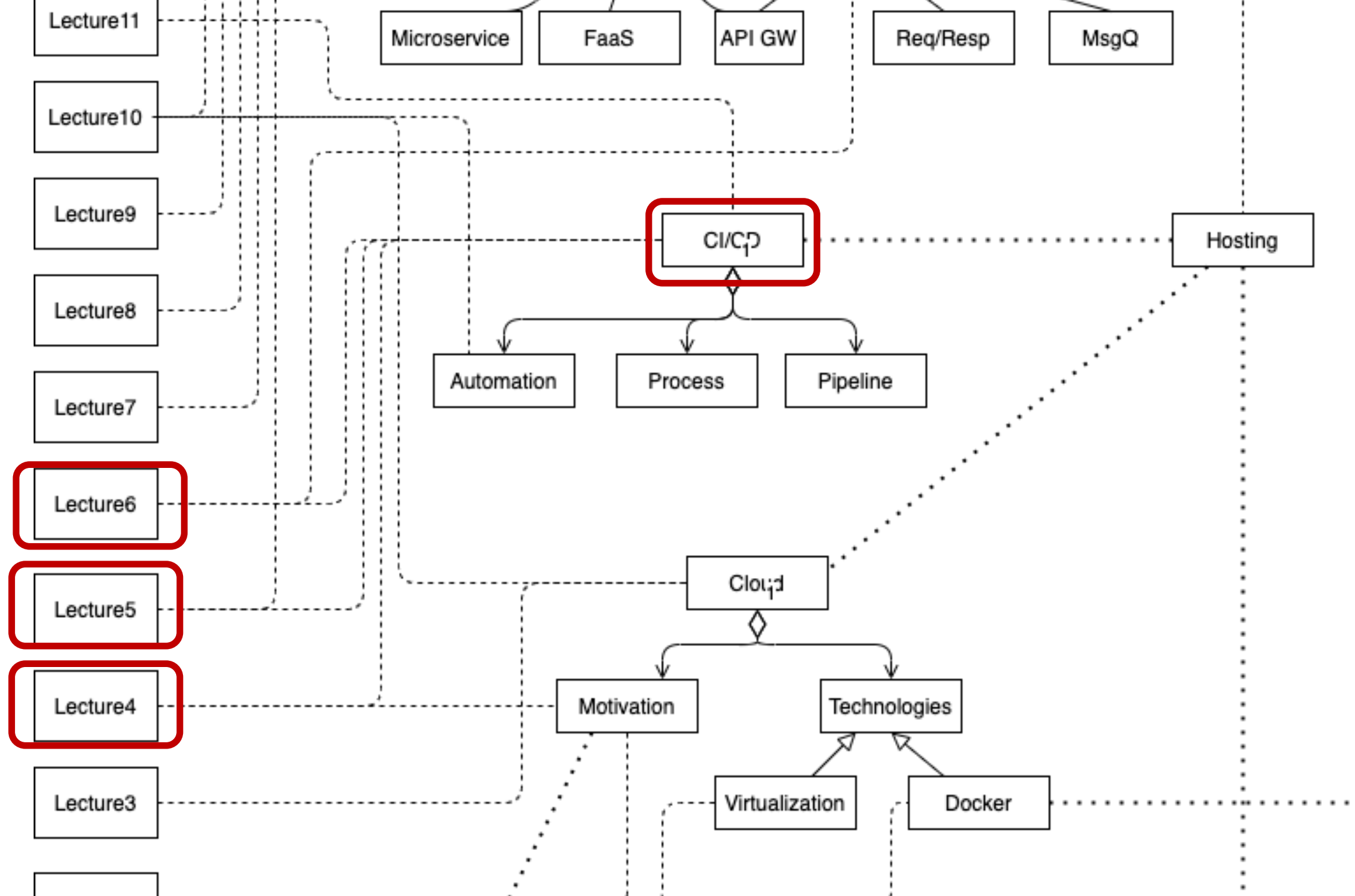


Docker compose



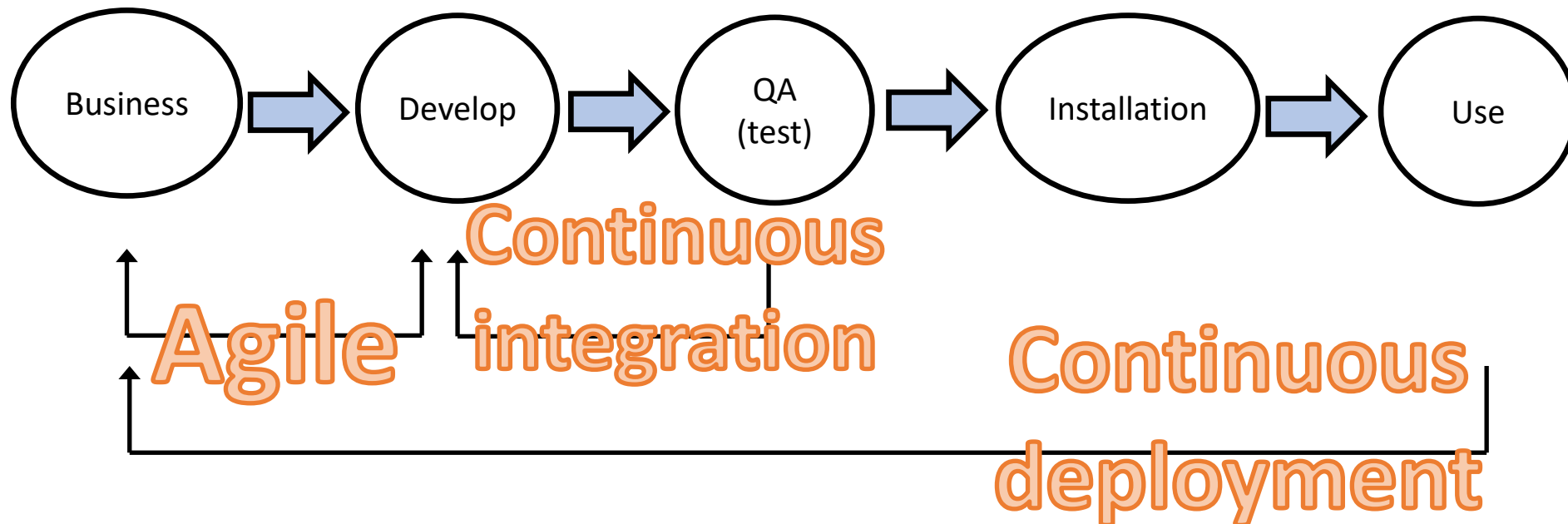
FIWARE platform architecture





Feedback in traditional development

(Case: Internet-based service; based on slide by Antti Tirilä)



Continuous delivery and deployment

(<http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>)

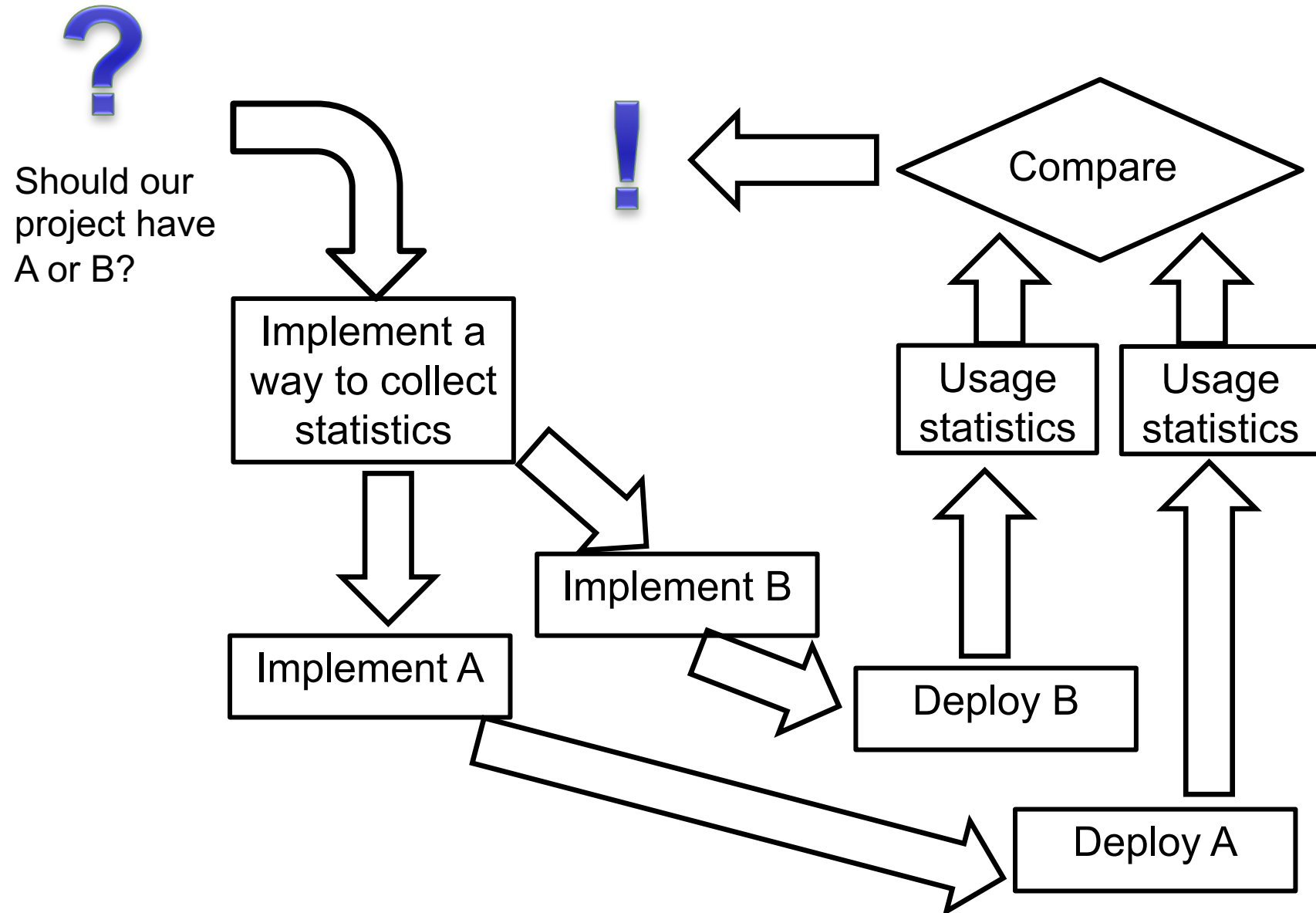
CONTINUOUS DELIVERY



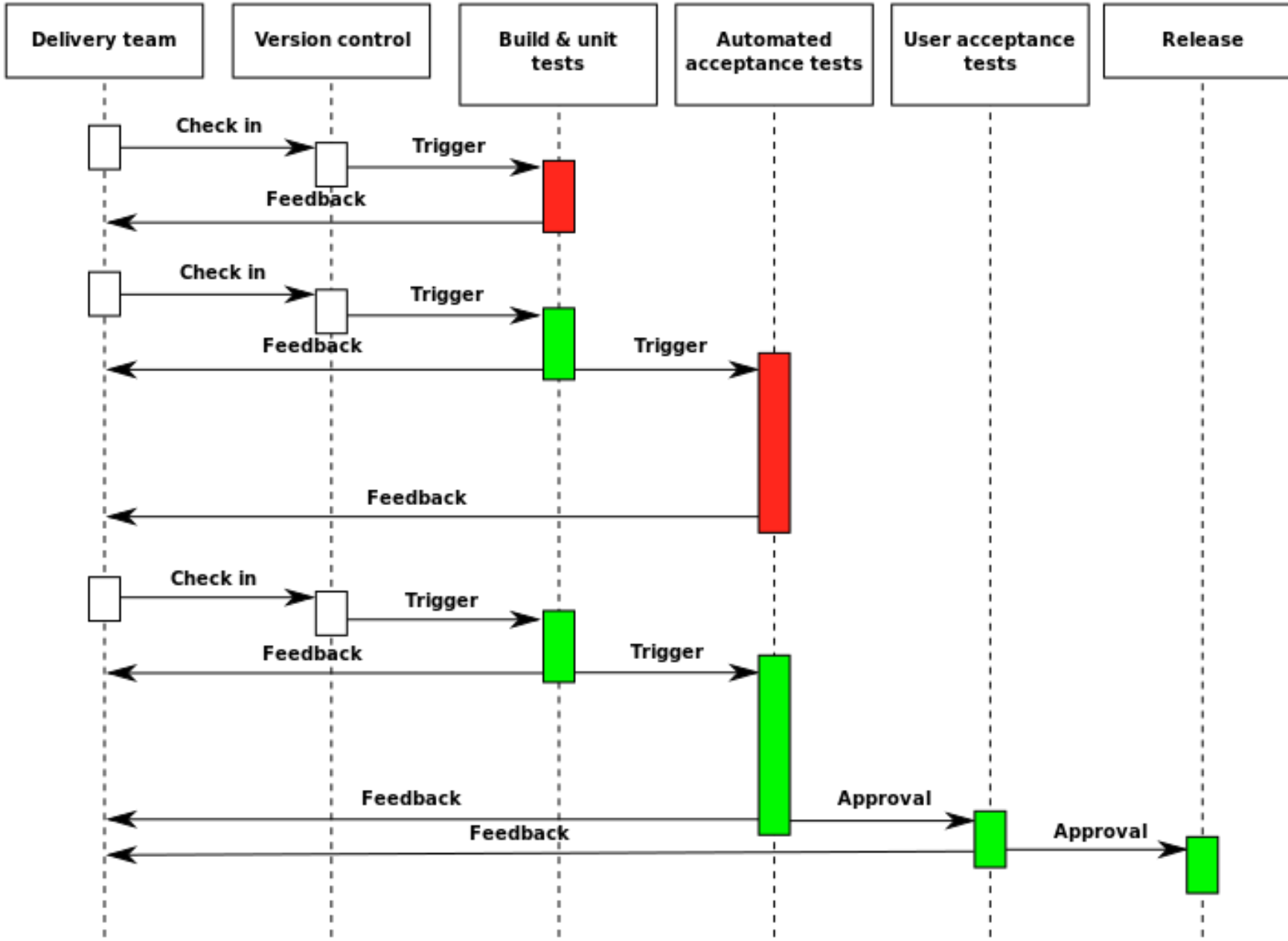
CONTINUOUS DEPLOYMENT



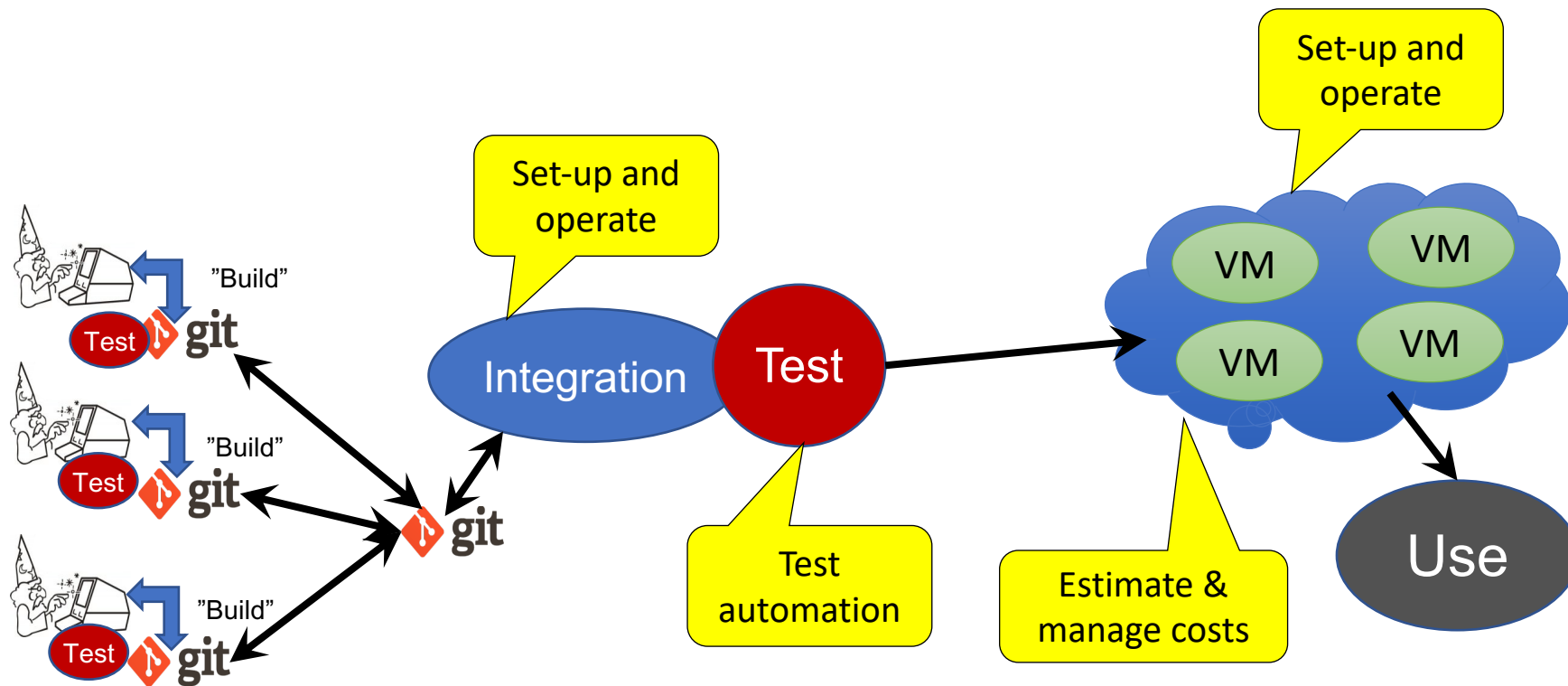
A/B Testing



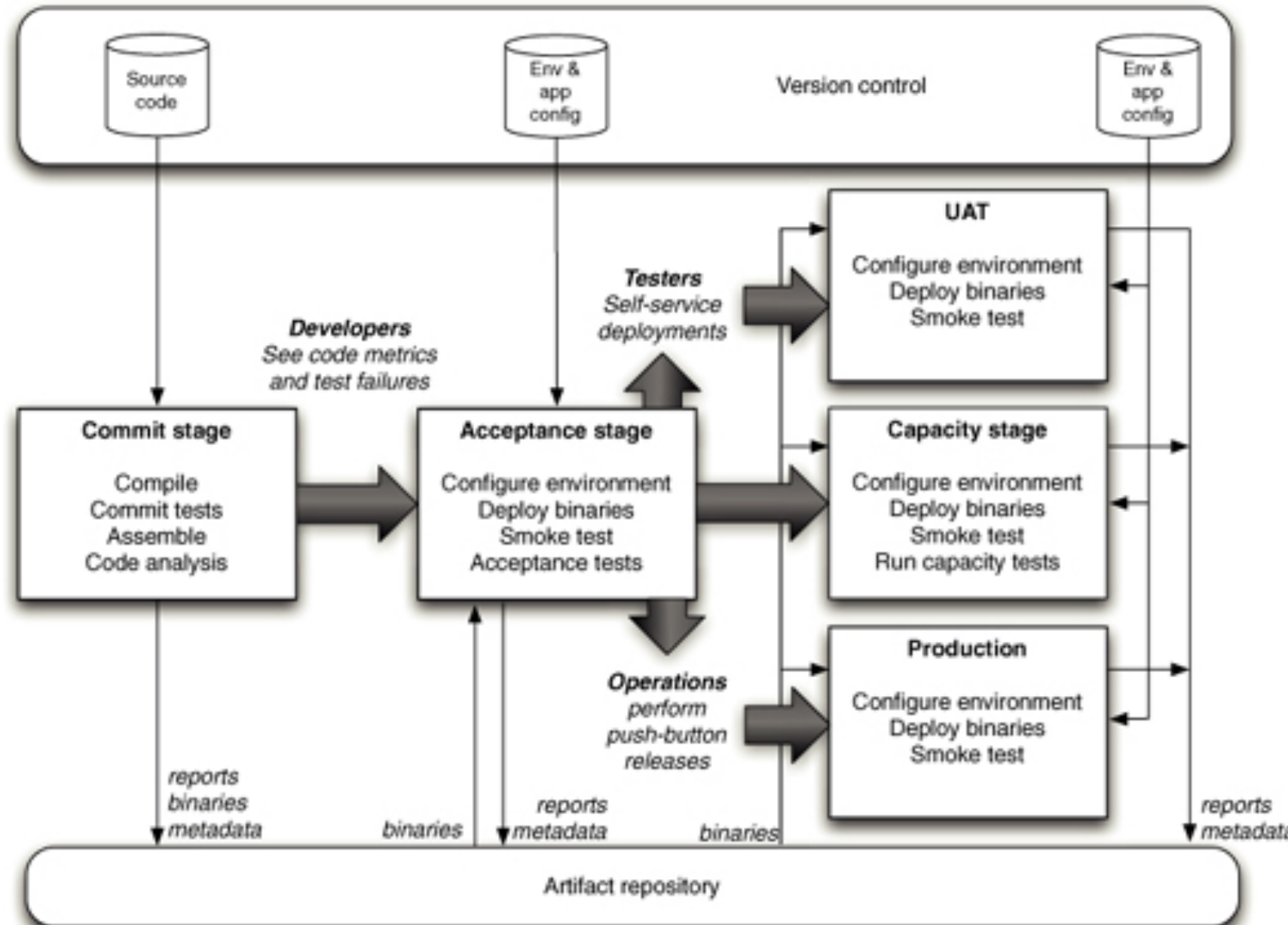
Deployment pipeline (a possible example)



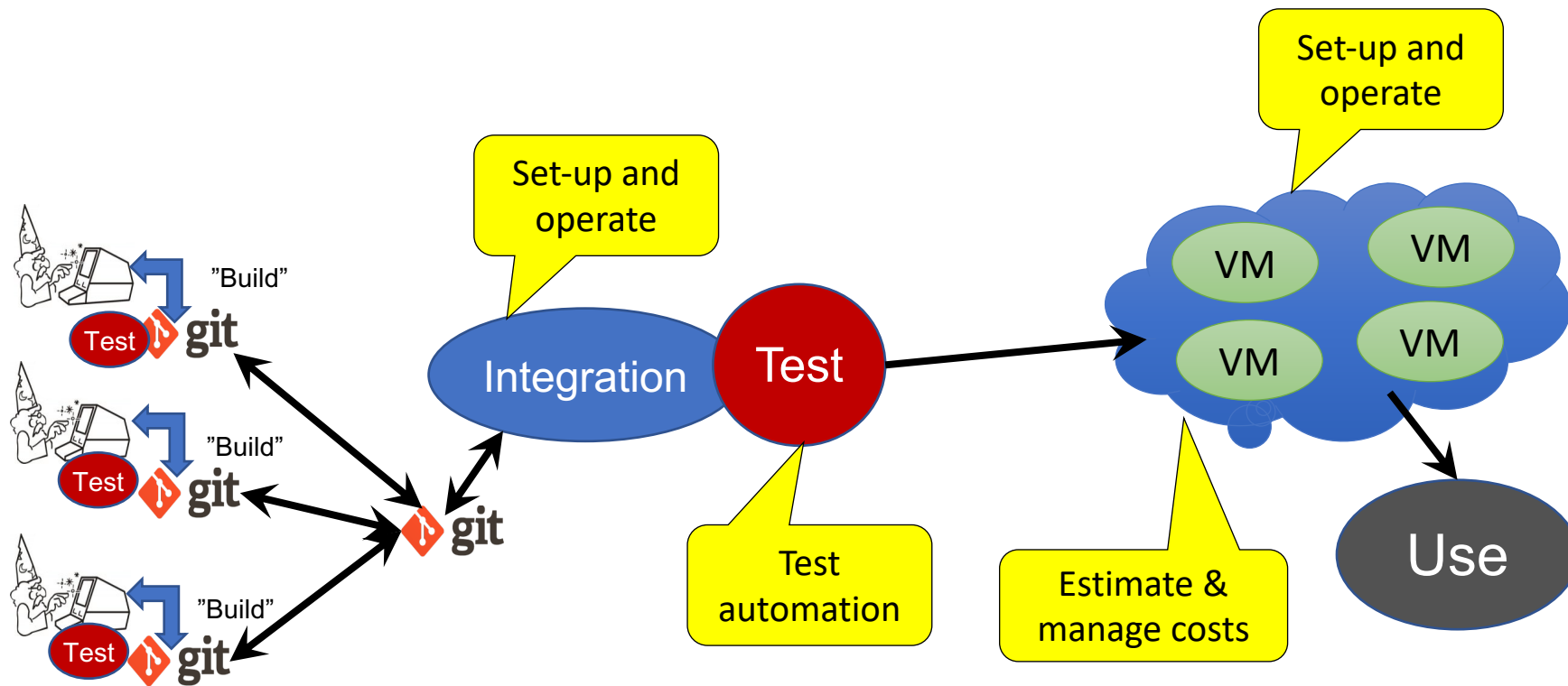
What does it really take to run CD?



Artifact repository



What does it really take to run CD?



CI – essential practices

(according to Humbley and Farley)

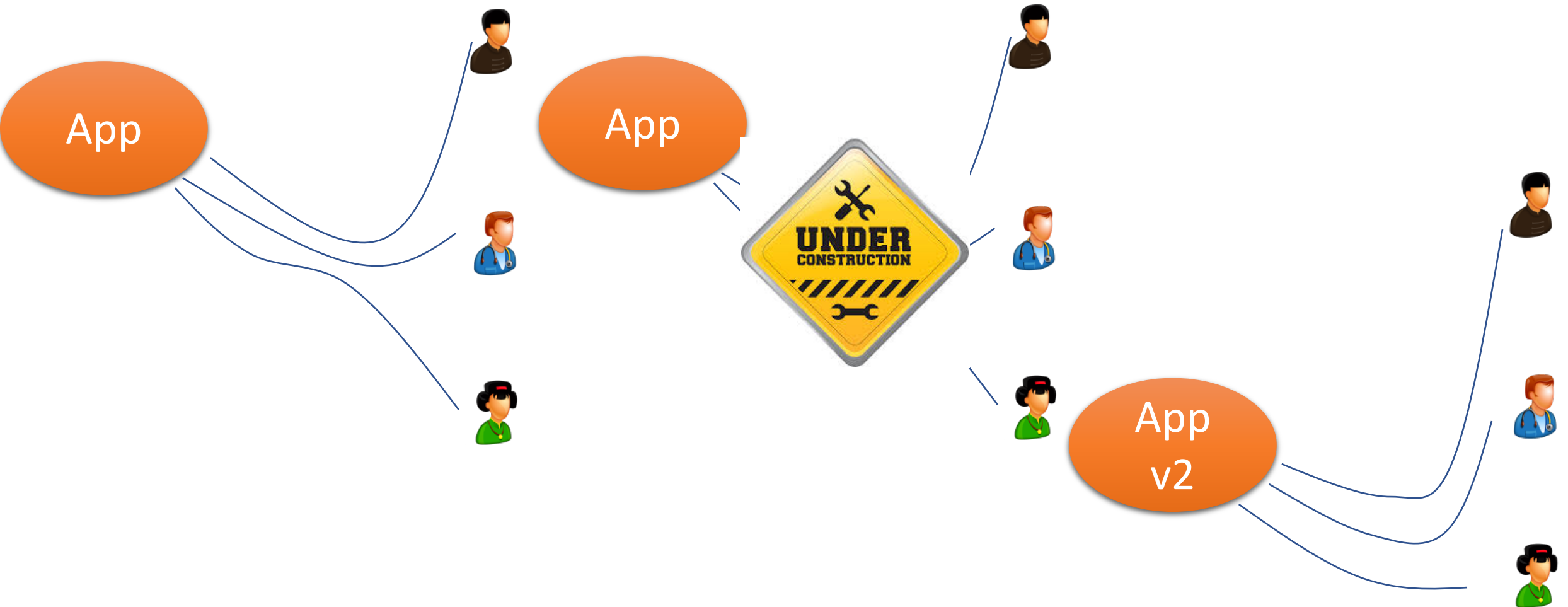
- Don't check in on a broken code
- Always run all commits tests locally before committing, or get your CI server to do it for you
- Wait for commit tests to pass before moving on
- Never go home on a broken build
- Always be prepared to revert to the previous revisions
- Time-box fixing before reverting
- Don't comment out failing tests
- Take responsible for all breakages that result from your changes
- Test-driven development

Deployment essential pract.

(according to Humbley and Farley)

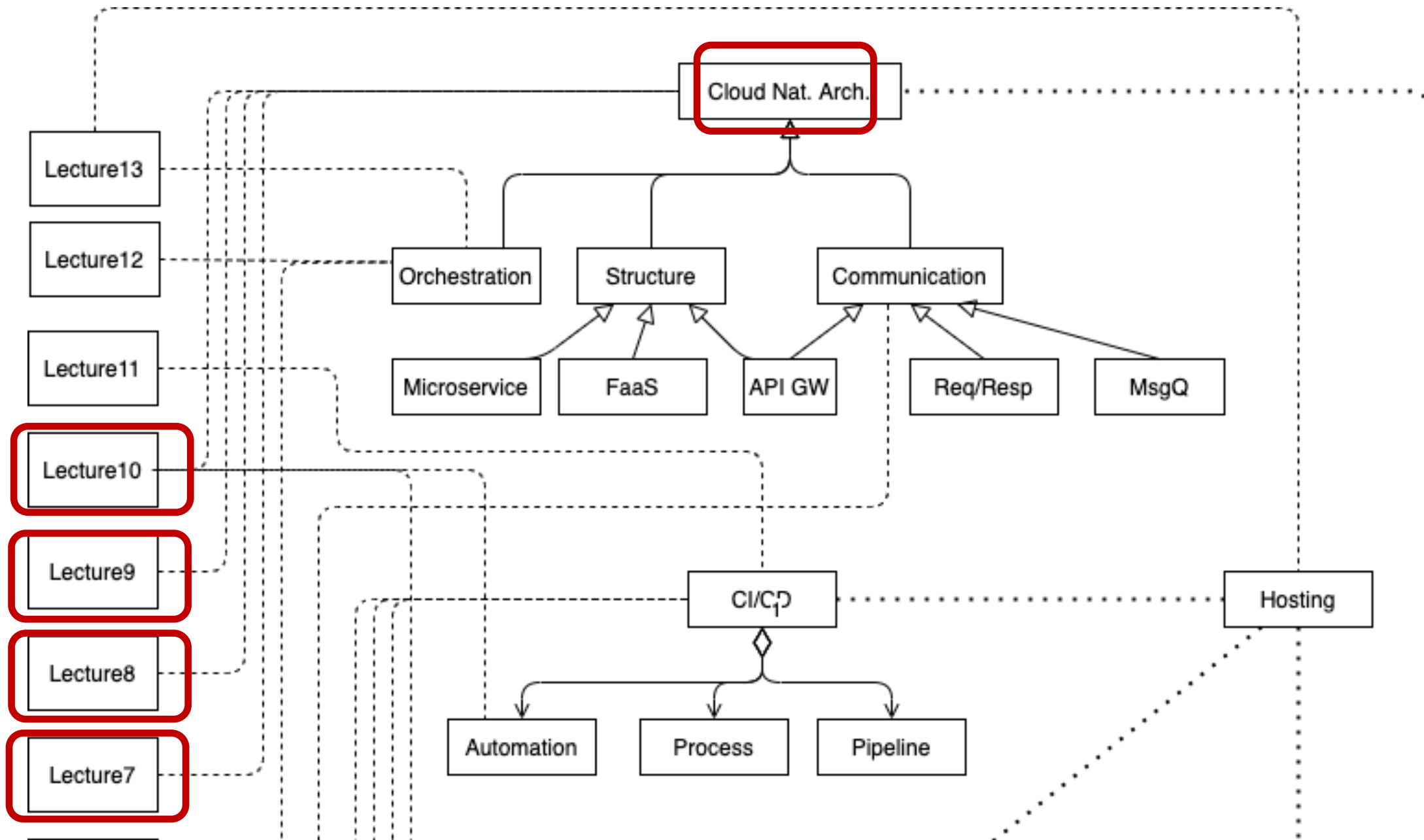
- Only build your binaries once
- Deploy the same way to every environment
- Smoke-test your deployments
- Deploy to copy of production
- Each change should propagate through the pipeline instantly
- If any part of pipeline fails, stop the line

A possible strategy to deploy a new version?



Deployment strategies

- Basic Deployment (aka Suicide) (<https://harness.io/2018/02/deployment-strategies-continuous-delivery/>) all nodes are updated at the same time
- Rolling Deployment (<https://harness.io/2018/02/deployment-strategies-continuous-delivery/>) nodes are updated incrementally,
- BlueGreenDeployment (<http://martinfowler.com/bliki/BlueGreenDeployment.html>) uses a router of incoming traffic as the tool. In this approach the new version (called green) is set up in parallel with the current (blue). When new (green) is ready, the router is switched to new (green) and blue is left as a backup. If something goes wrong with new, the router can be switched back to old - that means easy “rollback”.
- Canary Releases (<http://martinfowler.com/bliki/CanaryRelease.html>) implements the deployment incrementally. In this case the router first directs only part of the customers to the new version. If feedback is good, the other customers are moved to new version, too



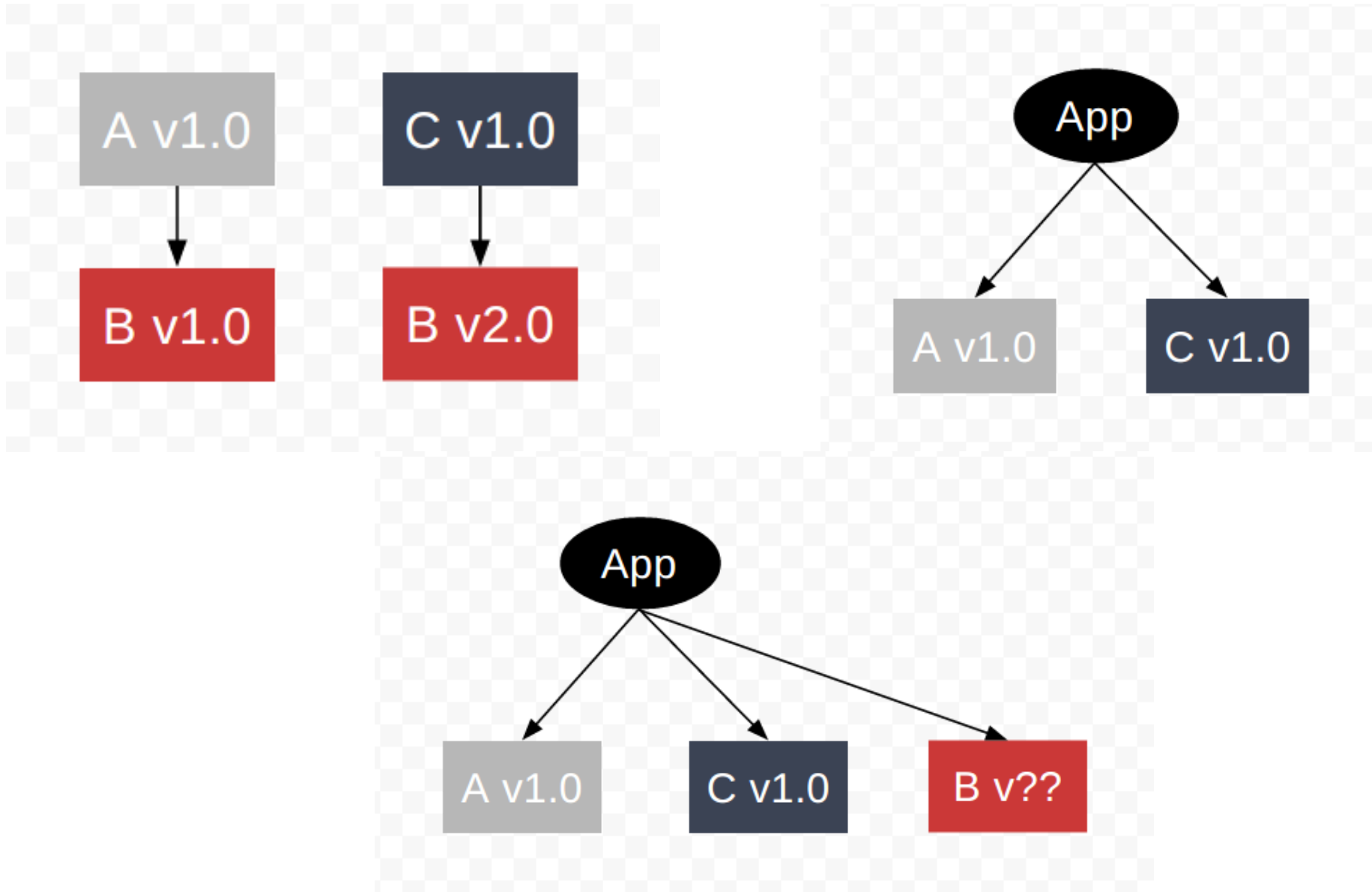
UNIX PROGRAM WRITTEN 30 YR AGO



JS PACKAGE WRITTEN 30 MONTHS AGO



<https://npm.github.io/how-npm-works-docs/theory-and-design/dependency-hell.html>



The old way

Static approach

- Libraries come with the compiler, or are installed to the development tool
- Compiler integrates application with libraries
- The integrated system is deployed to users

The Web & Cloud way

Dynamic approach

- Libraries are downloaded in a dynamic manner
- Huge number of libraries available, use each other, and are frequently updated (continuous delivery)
- npm
- pip

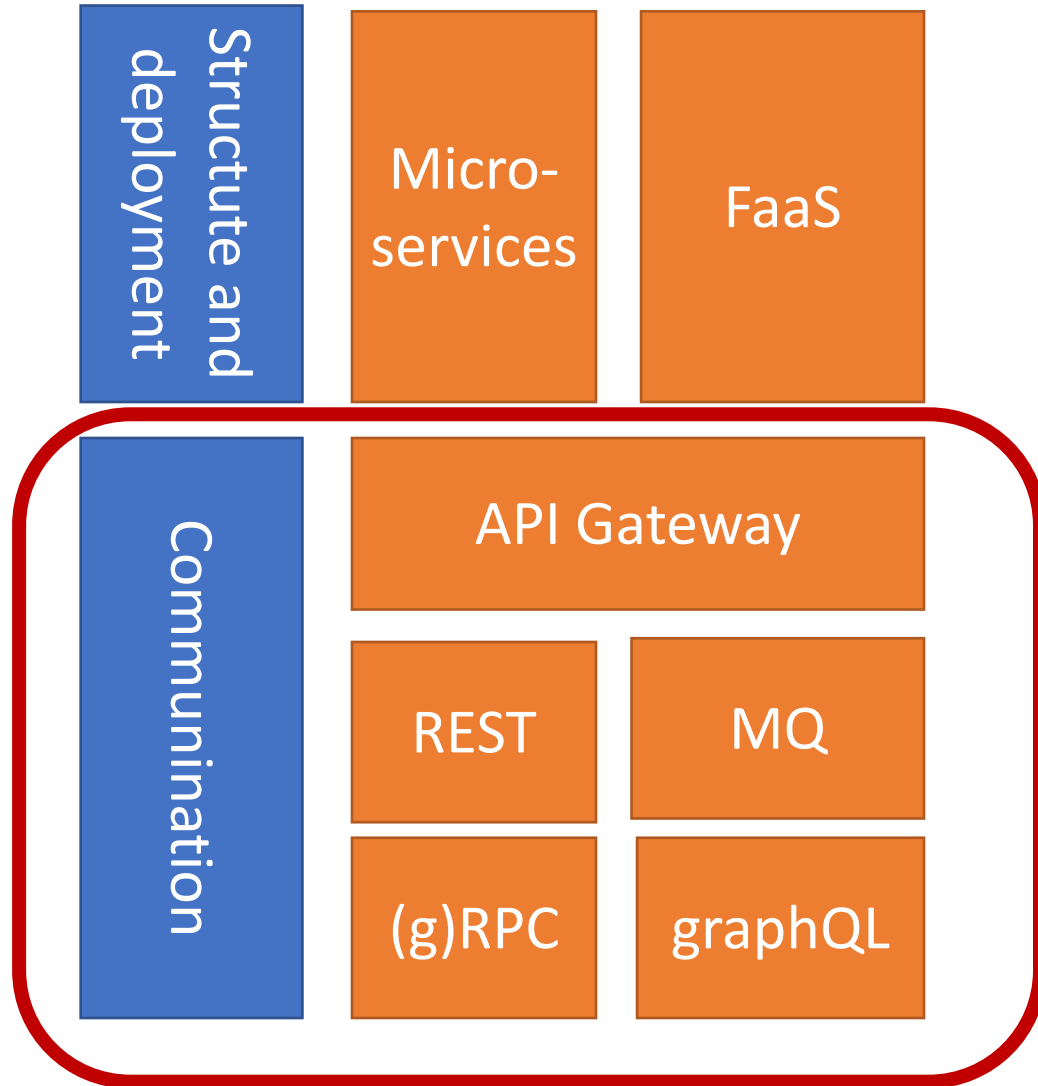
Package.json

```
{
  "name": "service1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "4.17.1",
    "request": "2.88.0"
  }
}
```

Dockerfile

```
#This is a sample Image  
FROM ubuntu  
MAINTAINER demouser@gmail.com  
RUN apt-get update  
RUN apt-get install -y nginx  
CMD ["echo","Image created"]
```

More about cloud-native architectures



Can be used in many ways

Designed for independent services

Standard ways to document
Designed for independent

Practically none on top of
Network infra

Practically none on
top of
Network infra

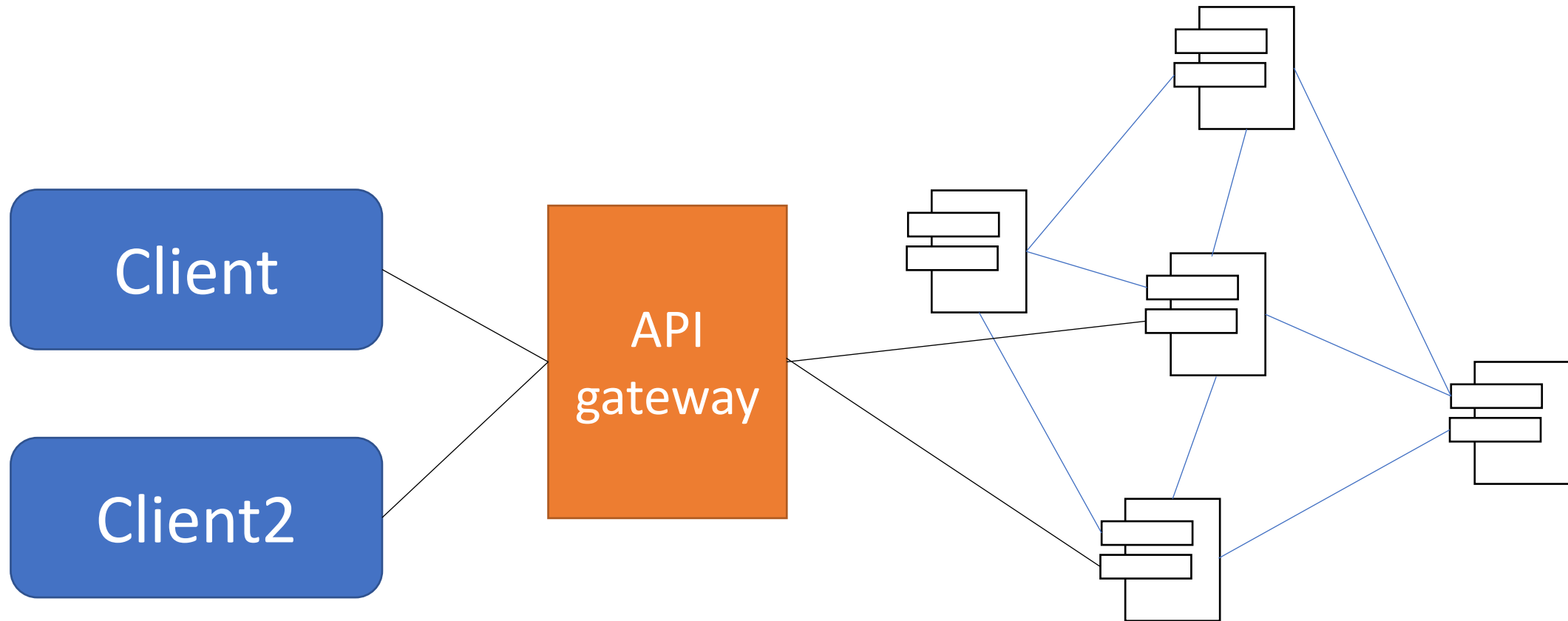
	Independent development	Independent deployment	Minimum central management
REST			
gRPC			
Message queue			

No standards: need to be
agreed on

The queue even supports
interrupts

The message queue need to
be maintained

How about external calls?



RECALL Interface segregation principle

“many client-specific interfaces are better than one general-purpose interface.”

“Make fine grained interfaces that are client specific”

“Clients should not be forced to depend upon methods they do not use”

- Big system with many dependencies = small change causes changed everywhere
- Large interfaces are split to smaller and role-base interfaces.
 - ⇒ changes do not affect everybody
 - ⇒ New features are easier to add
 - ⇒ Interfaces are easier to learn

Recap

David S. Linthicum, Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between, IEEE Cloud Computing, December 2017

- **Performance.** You'll typically be able to access provide better performance than is possible with nonnative features. For example, you can deal with an input/output (I/O) system that works with autoscaling and loadbalancing features.
- **Efficiency.** Cloud-native applications' use of cloud-native features and application programming interfaces (APIs) should provide more efficient use of underlying resources. That translates to better performance and/or lower operating costs.
- **Cost.** Applications that are more efficient typically cost less to run. Cloud providers send you a monthly bill based upon the amount of resources consumed, so if you can do more with less, you save on dollars spent.
- **Scalability.** Because you write the applications to the native cloud interfaces, you have direct access to the autoscaling and load-balancing features of the cloud platform.

the microservice architectural style is an approach to developing a single application as a **suite of small services** each **running in its own process** and **communicating with lightweight mechanisms**, often an HTTP resource API.

These services are built around **business capabilities** and **independently deployable** by fully **automated deployment machinery**.

There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

I. Nadareishvili et al., Microservice Architecture: Aligning Principles, Practices, and Culture, O'Reilly, 2016.

- small
- messaging enabled,
- bounded by contexts,
- **autonomously developed**,
- independently deployable,
- decentralized, and
- built and released with automated processes.

Some links

- 10 Key Attributes of Cloud-native Applications, <<https://thenewstack.io/10-key-attributes-of-cloud-native-applications/>>
- What are cloud-native applications?

1. Packaged as lightweight containers
2. Developed with best-of-breed languages and frameworks
3. Designed as loosely coupled microservices
4. Centered around APIs for interaction and collaboration
5. Architected with a clean separation of stateless and stateful services
6. Isolated from server and operating system dependencies
7. Deployed on self-service, elastic, cloud infrastructure
8. Managed through agile DevOps processes
9. Automated capabilities
10. Defined, policy-driven resource allocation

ops>

id-

n: The
ecember

**Do you really want to keep
your containers running all the time
if you need to pay for it?**

**Do you really want to operate
and maintain your containers –
your developers could also
do something else.**

Serverless computing

Baldini et al: Serverless Computing:

Current Trends and Open Problems, Research Advances in Cloud Computing, Springer, 2017.

A cloud-native platform

for

- short-running, stateless computation
- event driven applications

which

- scale up and down instantly and automatically
- and
- charge for actual usage and high granularity

Stateful vs stateless computation

- If a service has an internal state it is difficult to
 - Scale it
 - Move it to other server or other hosting system

=> Stateless Services are subject to cloud-specific optimizations
- The internal state may be
 - volatile or
 - non-volatile
 - ... in memory, file local to container,
- Serverless / FaaS

7R's of cloud Migration

Replace

with imilar or
improved
but SaaS

Reuse

in the new SaaS
version

Refactor

towards cloud-
native
architecture

Replatform

by using cloud
services

Rehost

to a VM

Retain

Retire

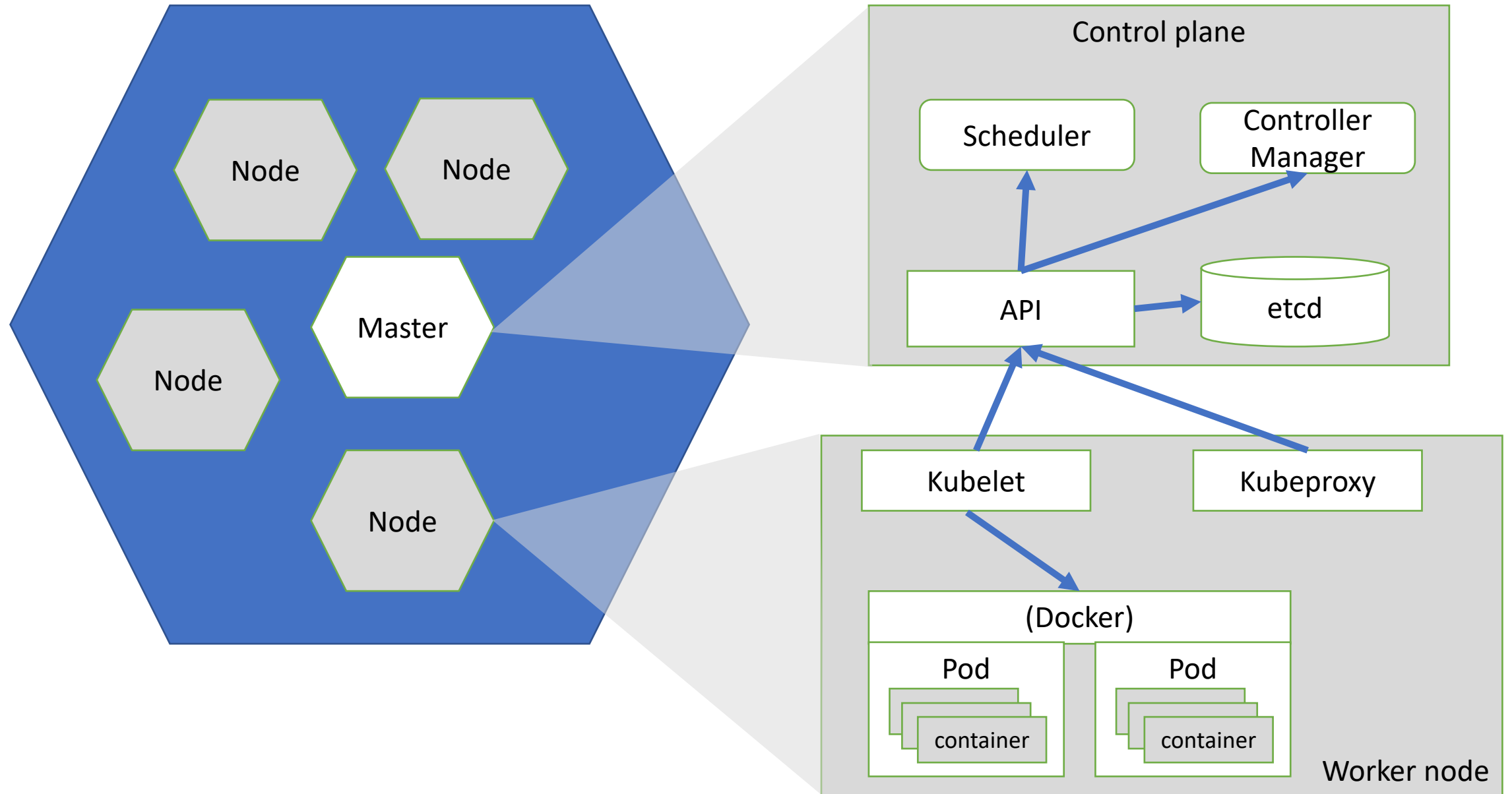
Infrastructure as code

From: <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code>

Infrastructure as Code (IaC) is

- the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model,
- using the same versioning as DevOps team uses for source code.
- Like the principle that the same source code generates the same binary, an IaC model generates the same environment every time it is applied.
- IaC is a key DevOps practice and is used in conjunction with [continuous delivery](#).

Kubernetes Kluster



Alternative approaches to define deployment

- Set-up everything when the image is created
 - Very static
- Make the container to auto-update
 - You need to know in advance what might change
- ~~Put stuff to shared folder (use volume)~~
- Use configuration tools
 - Work also for full virtual machines and computers

How prepare to exam

- Lecture videos & slides
- Read
 - - Peter Mell; Timothy Grance (September 2011). The NIST Definition of Cloud Computing (Technical report). National Institute of Standards and Technology: U.S. Department of Commerce. doi:10.6028/NIST.SP.800-145. Special publication 800-145. <<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>>
 - - 10 Key Attributes of Cloud-native Applications, <<https://thenewstack.io/10-key-attributes-of-cloud-native-applications/>>
 - - What are cloud-native applications? <<https://opensource.com/article/18/7/what-are-cloud-native-apps>>
 - - M. Leppänen et al., "The highways and country roads to continuous deployment," in IEEE Software, vol. 32, no. 2, pp. 64-72, Mar.-Apr. 2015. doi: 10.1109/MS.2015.50, URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7057604&isnumber=7057030>
 - - Continuous Delivery web page: <https://continuousdelivery.com>
 - - Lwakatare , Kilamo , Karvonen, Sauvola , Heikkilä, Itkonen, Kuvaja, Mikkonen, Oivo & Lassenius: DevOps in practice : A multiple case study of five companies, Information and Software Technology , vol. 114 , pp. 217-230 . <https://doi.org/10.1016/j.infsof.2019.06.010>
 - <https://www.wired.com/insights/2011/12/service-level-agreements-in-the-cloud-who-cares>