

Lecture 08

Cloud Native – part 2

Kari Systä
20.10.2020

Schedule for coming weeks

Week	Lecture	Plussa exercises (deadlines)
8/43	20.10 Cloud-native architectures part 2	19.10 Next exercise closes 25.10 Project instructions opens
9/44	27.10 Introduction to project, Gitlab CI	
10/45	01.11 Testing and testing automation	
11/46	10.11 Guest Lecture, CD pipeline at cargotec	
12/47	17.11	
13/48	24.11	
14/40	01.12	

Course practicalities

- Cloud exercise was returned by 70
- Compose exercise was returned by 52+4
- Message-queue communication has been returned by 42 so far.
- I have now checked all the compose-exercises and will give feedback in Plussa in coming days.
- A few notes
 - Clearly over half of systems worked for me “as such”.
 - Surprisingly many students used **volume** to make application code visible – in addition to installing it. This is obviously wrong.
 - Seems that many systems based on Microsoft technologies are probably created with some IDE. => a lot of “boilerplate code”.
 - With some libraries, e.g. Flask, it is difficult to fulfil the criteria.
 - We are on a way to automatize the first-line checking => please follow the instructions carefully

A job opportunity

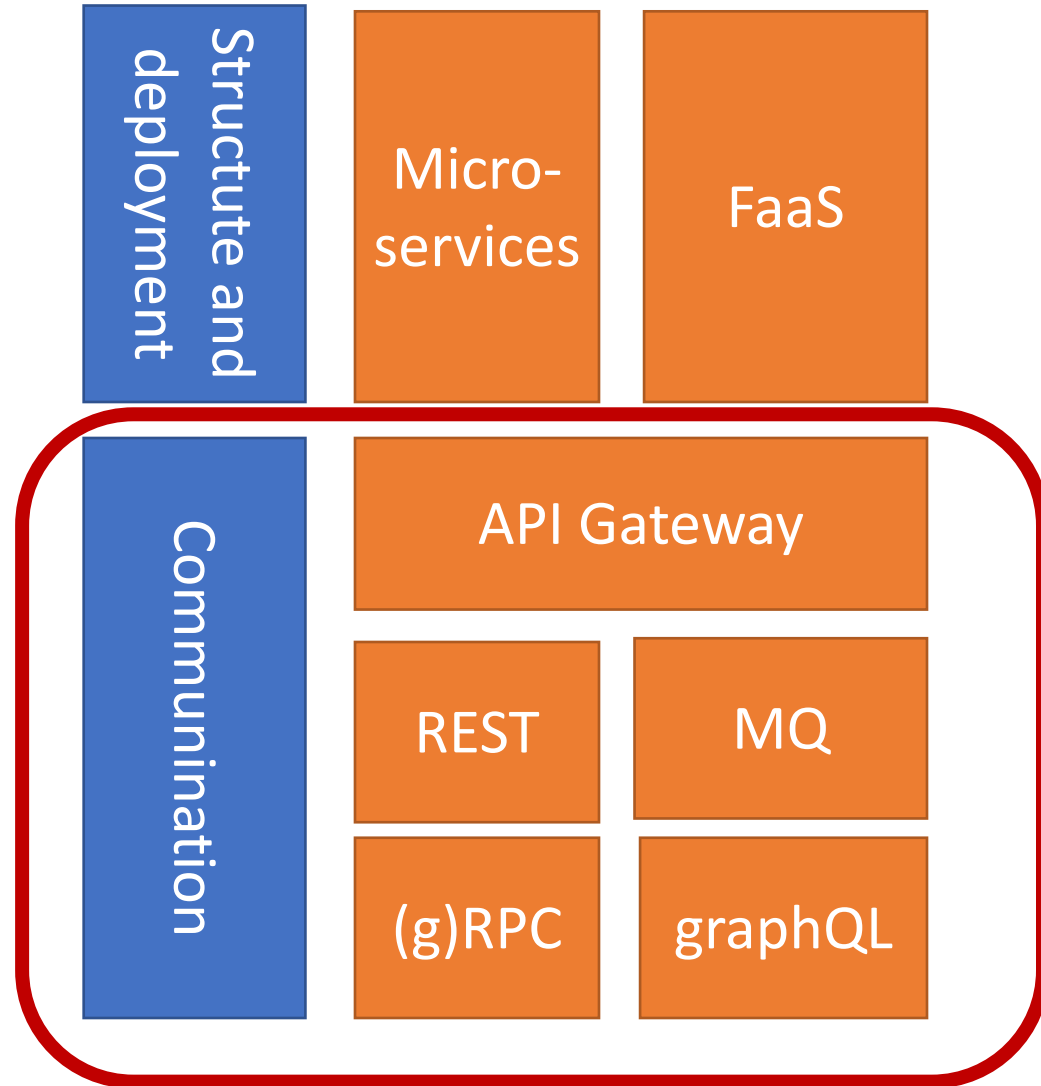
“I have a position for an MSc. Thesis:

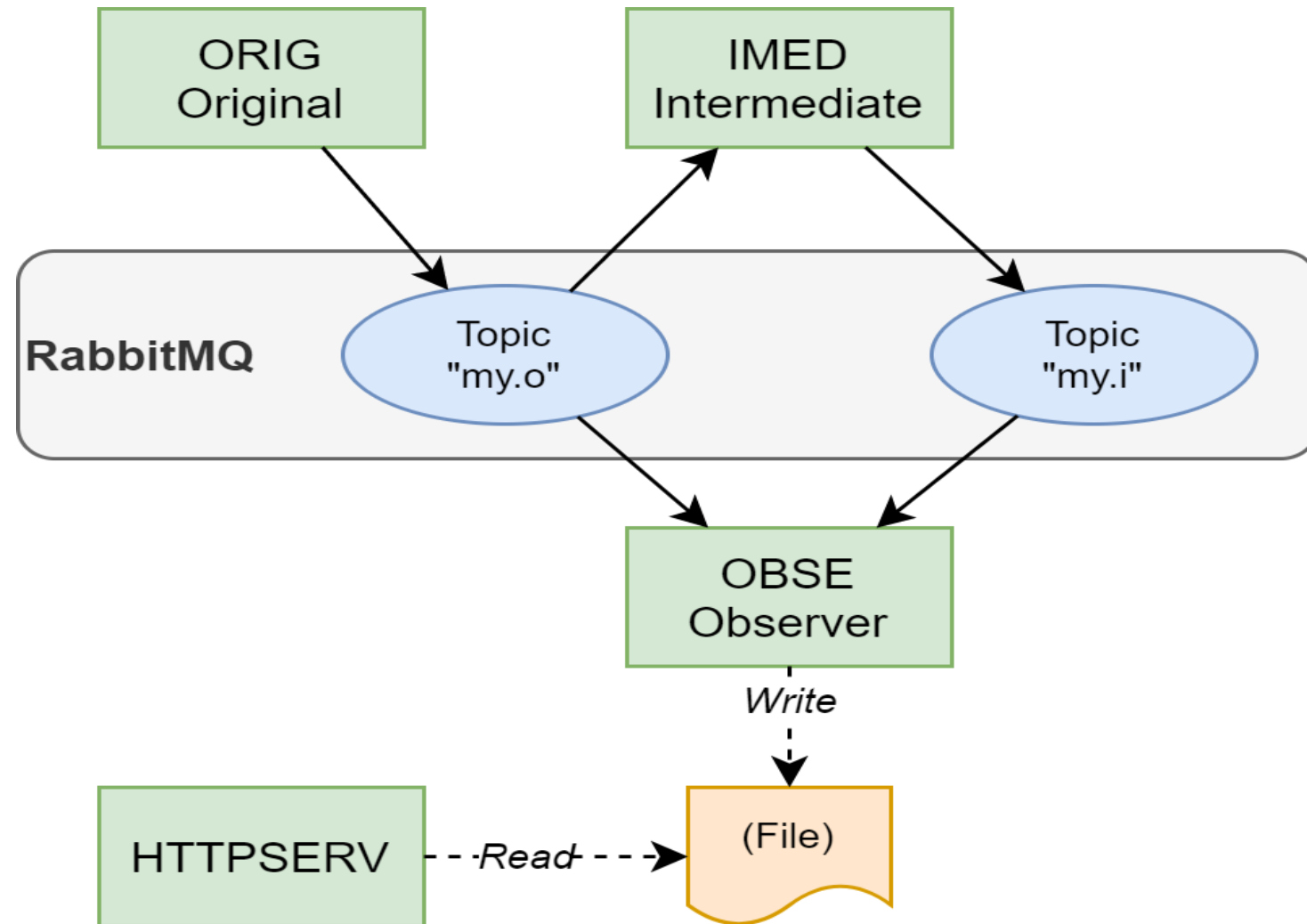
- <https://ats.talentadore.com/apply/Master%27s-Thesis-Worker%2C-Web-based-Graphical-Programming-Environment-for-Connected-Factories/ZQxVW8>

If you have some student you would like to refer, please let me know.

”

More about cloud-native architectures





WHAT IS ESSENTIAL OF REST



Architectural principles of REST

- **Client-server architecture**
- **Statelessness**
 - Everybody gets same answer
 - Repeated operation (GET, PUT) does not have an effect
- **Cacheability**
 - For performance and scalability
- **Layered system**
 - Allows proxies etc
- **Uniform interface**

Uniform interface

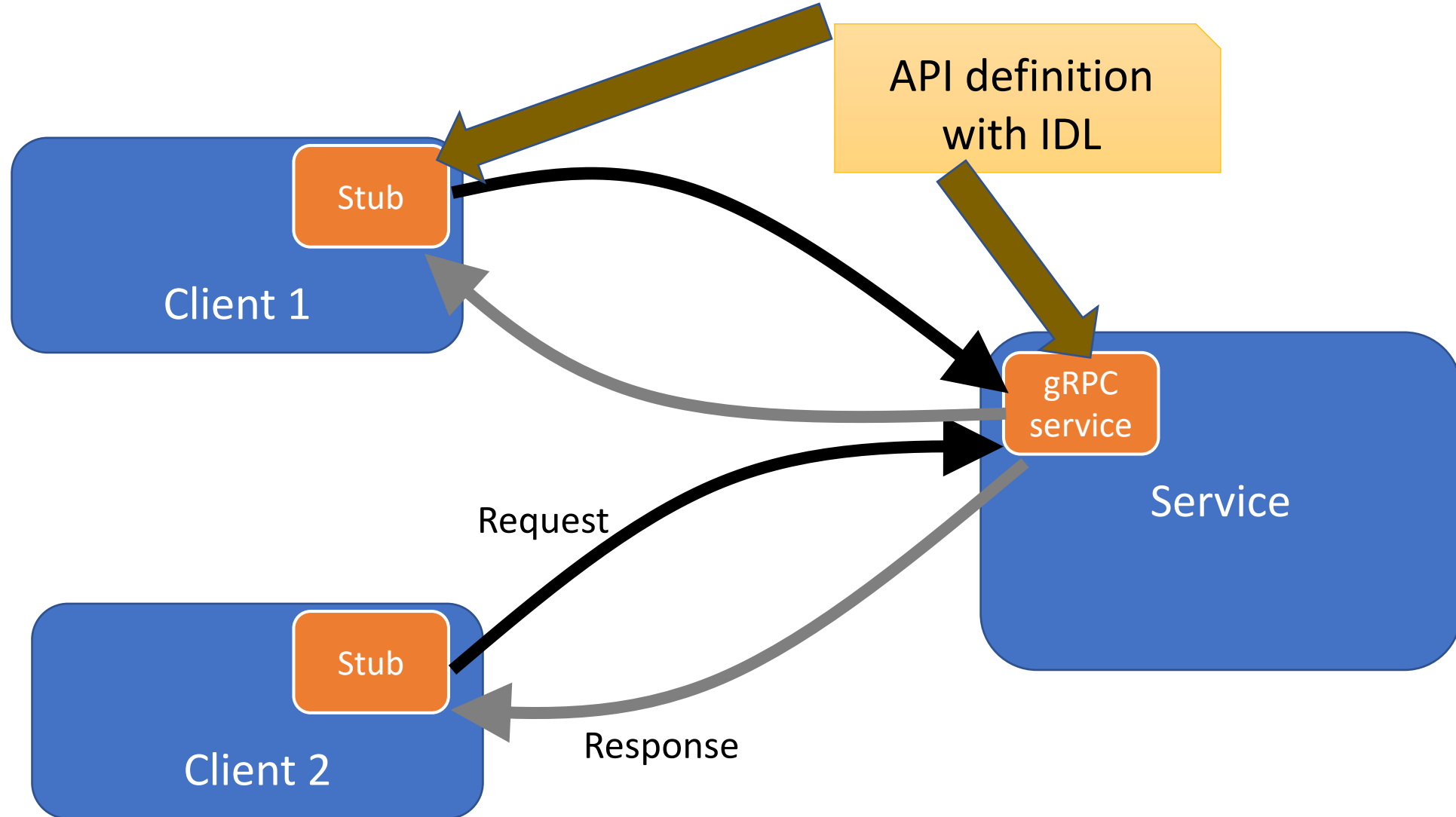
- Everything is a resource that is fetched, modified, created, deleted
 - CRUD = CREATE, READ, UPDATE, DELETE
 - HTTP verbs: GET, PUT, POST, DELETE
 - Resource manipulation through representations
- Resource identification in requests
 - URIs
 - Separated from representation (XML, JSON,...)
 - MIME-types
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

But the "calls" can be laborious

```
let message = "Hello from " + req.client.remoteAddress + ":" +  
req.client.remotePort + " to " + req.client.localAddress + ":" +  
req.client.localPort;  
  
request('http://server2:4000/getServer', { json: true },  
  (err, response, body) => {  
    if (err) {  
      return console.log(err);  
    }  
    res.send(message + " " + body); })
```

REST vs RPC

gRPC – RPC over HTTP



Example API description

```
service Greeter {  
    // Sends a greeting  
    rpc SayHello (HelloRequest) returns (HelloReply) {}  
    // Sends another greeting  
    rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}  
}  
  
// The request message containing the user's name.  
message HelloRequest { string name = 1; }  
  
// The response message containing the greetings message  
HelloReply { string message = 1; }
```

Call in JavaScript and Python

```
function main() {  
  var client = new hello_proto.Greeter('localhost:50051',  
                                       grpc.credentials.createInsecure());  
  client.sayHello({name: 'you'}, function(err, response) {  
    console.log('Greeting:', response.message);  
  });  
  client.sayHelloAgain({name: 'you'}, function(err, response) {  
    console.log('Greeting:', response.message);  
  });  
}
```

```
def run():  
    channel = grpc.insecure_channel('localhost:50051')  
    stub = helloworld_pb2_grpc.GreeterStub(channel)  
    response = stub.SayHello(helloworld_pb2.HelloRequest(name='you'))  
    print("Greeter client received: " + response.message)  
    response = stub.SayHelloAgain(helloworld_pb2.HelloRequest(name='you'))  
    print("Greeter client received: " + response.message)
```

And C++

```
std::string SayHelloAgain(const std::string& user) {  
    // Follows the same pattern as SayHello.  
    HelloRequest request;  
    request.set_name(user);  
    HelloReply reply;  
    ClientContext context;  
  
    // Here we can use the stub's newly available method we just added.  
    Status status = stub_->SayHelloAgain(&context, request, &reply);  
    if (status.ok()) {  
        return reply.message();  
    } else {  
        std::cout << status.error_code() << ": " << status.error_message()  
                  << std::endl;  
        return "RPC failed";  
    }  
}
```

Message-bus instead of HTTP

- Challenges: increased network operations, tight service coupling
- Message bus helps to define how services communicate, service discovery reduces operational complexity
- Asynchronous messaging leads to
 - loosed coupling
 - More complex logic (async a cousin of parallelism)
- Actually, there are multiple options
 - RPC, REST, Asynchronous message, application-specific protocols

What is the main difference between RPC and REST?

Consequences

	Independent development	Independent deployment	Minimum centralized management
REST			
gRPC			
Message queue			

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

Let's have another thinking
exercise

GraphQL(examples from

<https://medium.com/tech-tajawal/backend-for-frontend-using-graphql-under-microservices-5b63bbfcd7d9>)

- REST request

GET <http://127.0.0.1/api/accounts>

- Response

```
[
  {
    "id": 88,
    "name": "Mena Meseha",
    "photo": "http://..m/photo.jpg"
  },
  ...
]
```

- GraphQL request

POST <http://127.0.0.1/graphql>

- Payload

query {accounts {id, name, photo}}

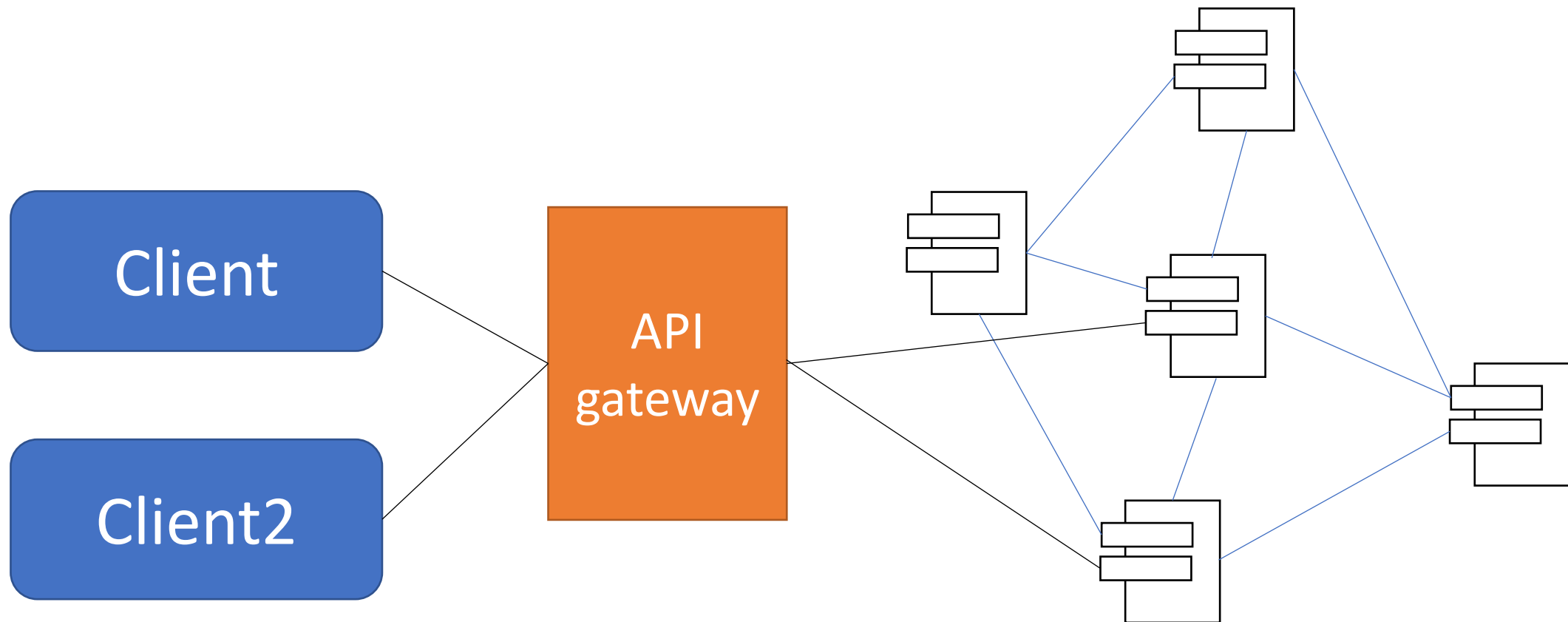
- Response

```
{
  "data": {
    "accounts": [ {
      "id": 88,
      "name": "Mena Meseha",
      "photo":
        "http://...com/photo.jpg"
    },
    ...
  ]
}
```

Let's analyze some claims of the previous source

- **1. Data Acquisition:** REST lacks scalability and GraphQL can be accessed on demand. The payload can be extended when the GraphQL API is called.
- **2. API calls:** REST's operation for each resource is an endpoint, and GraphQL only needs a single endpoint, but the post body is not the same.
- **3. Complex data requests:** REST requires multiple calls for nested complex data, GraphQL calls once, reducing network overhead.
- **4. Error code processing:** REST can accurately return HTTP error code, GraphQL returns 200 uniformly, and wraps error information.
- **5. Version number:** REST is implemented via v1/v2, and GraphQL is implemented through the Schema extension.

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

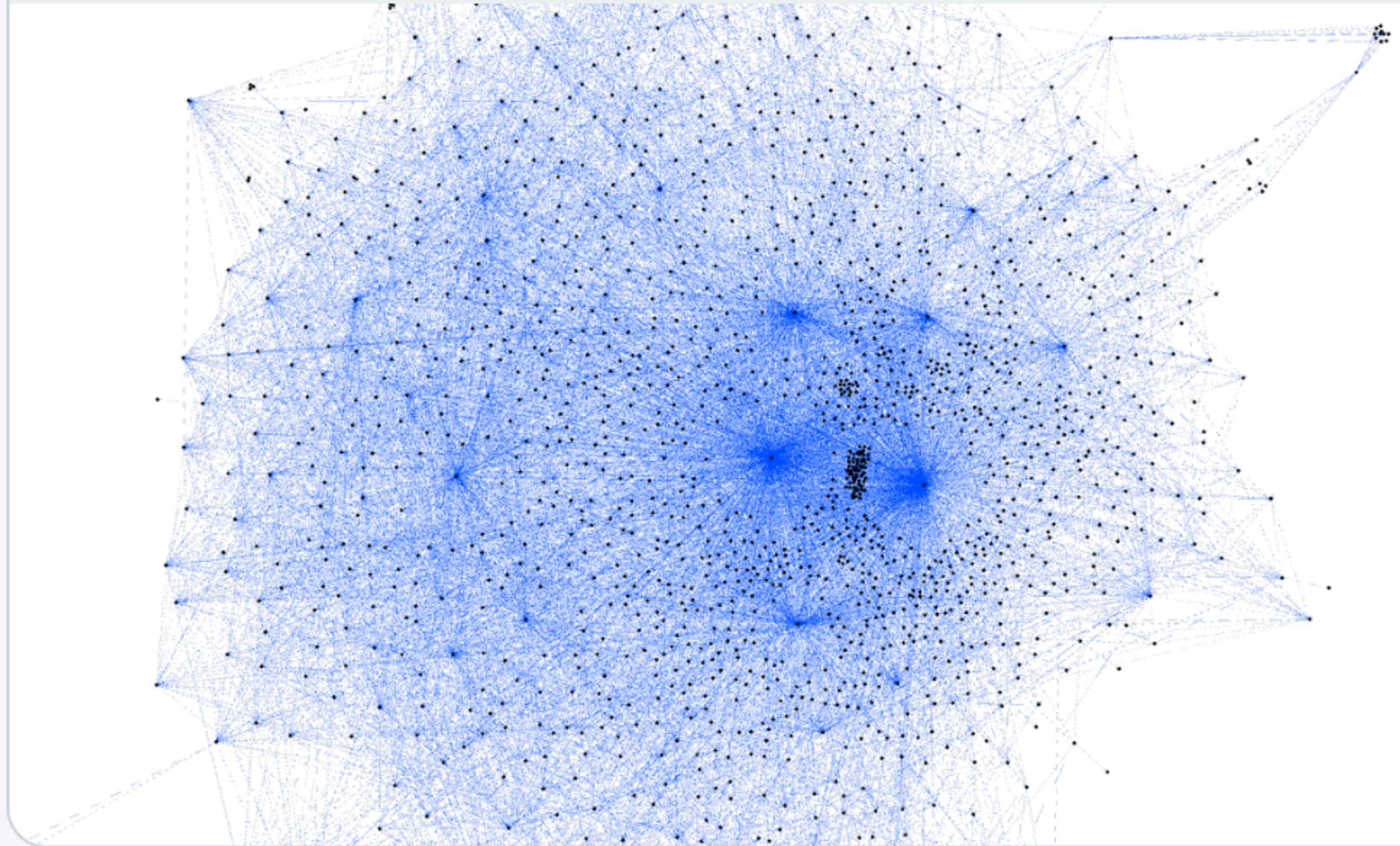
From Twitter



Jack Kleeman @JackKleeman · Nov 1

1500 microservices at @monzo; every line is an enforced network rule allowing traffic

[Show this thread](#)



- <https://microservices.io/patterns/apigateway.html>
- <https://whatis.techtarget.com/definition/API-gateway-application-programming-interface-gateway>

API gateway pattern

<https://microservices.io/patterns/apigateway.html>

Problem

- How do the clients of a Microservices-based application access the individual services?

Forces

- The granularity of APIs provided by microservices is often different than what a client needs and too fine grained.
- Different clients need different data.
- Network performance is different for different types of clients.
- Partitioning into services can change over time and should be hidden from clients
- Services might use a diverse set of protocols, some of which might not be web friendly

Solution

- Implement an API gateway that is the single entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.

Other patterns

Application architecture patterns

- Which architecture should you choose for an application?

Decomposition

- How to decompose an application into services?

Data management

- How to maintain data consistency and implement queries?

Transactional messaging

- How to publish messages as part of a database transaction?

Testing

- How to make testing easier?

Deployment patterns

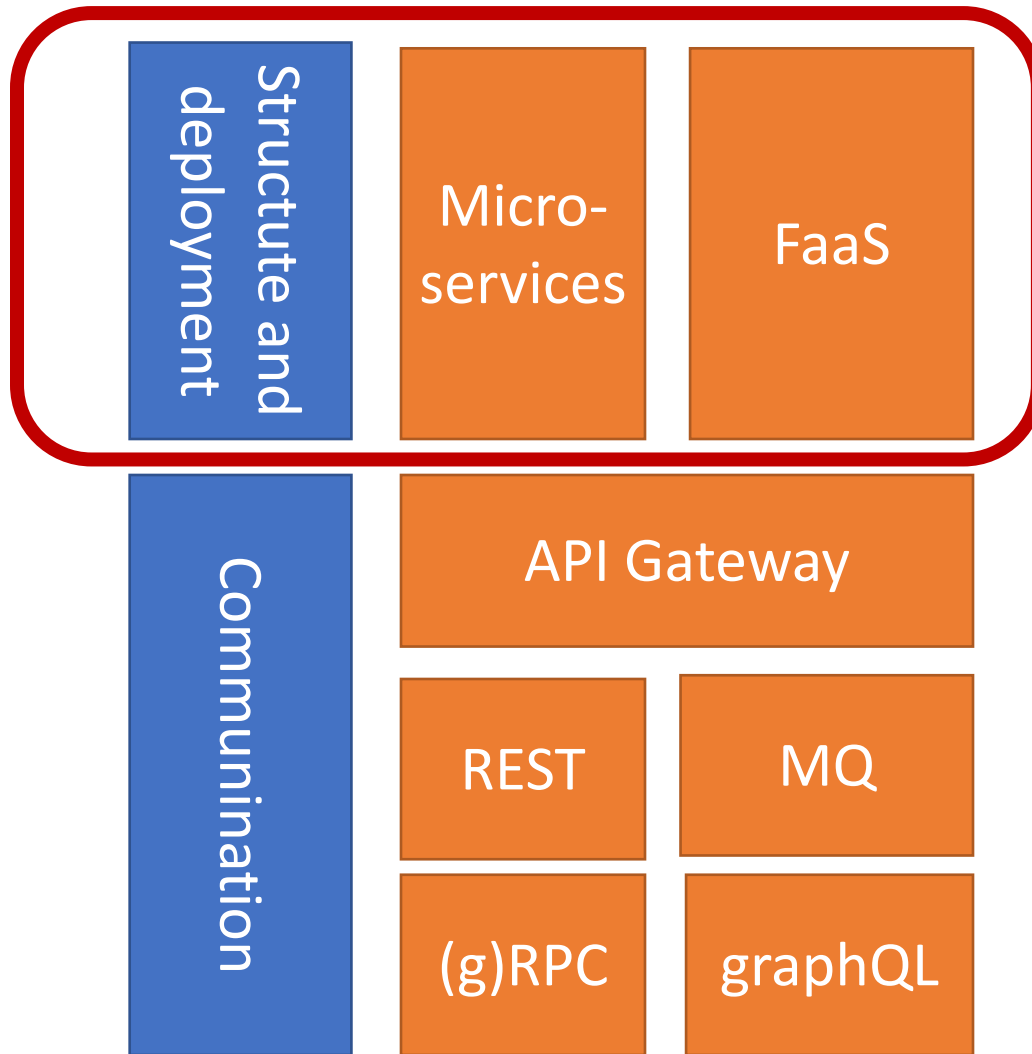
- How to deploy an application's services?

Cross cutting concerns

- How to handle cross cutting concerns?

Communication patterns

More about cloud-native architectures



Cloud-native applications and architectures

Some definitions

- If an app is "cloud-native," it's specifically designed to provide a consistent development and automated management experience across private, public, and hybrid clouds.
- A native cloud application (NCA) is a program that is designed specifically for a cloud computing architecture.
NCAs are designed to take advantage of cloud computing frameworks, which are composed of loosely-coupled cloud services. That means that developers must break down tasks into separate services that can run on several servers in different locations. Because the infrastructure that supports a native cloud app does not run locally, NCAs must be planned with redundancy in mind so the application can withstand equipment failure and be able to re-map IP addresses automatically should hardware fail.

Some links

- 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>>
- Native cloud application (NCA),
<<https://searchitoperations.techtarget.com/definition/native-cloud-application-NCA>>
- Understanding cloud-native applications,
<<https://www.redhat.com/en/topics/cloud-native-apps>>
- David S. Linthicum, Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between, IEEE Cloud Computing, December 2017.

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

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.

Build around business capabilities?

- A way to split monolith to micro services
- Traditional SOA way is to look at static or dynamic dependencies
 - And minimize inter-service calls
- Should support independent development:
 - *Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.*
 - -- Melvyn Conway, 1967
- Should support independent deployment
 - Testing
 - Timing

Projects ▾ Groups ▾ Activity Milestones Snippets ▾ Search or jump to...

Faculty of Information Technology and Communication Sciences > ... > TIE-23536 > plusa-syksy2019 > Pipelines

All 91 Pending 0 Running 0 Finished 91 Branches Tags

Run Pipeline

Status	Pipeline	Triggerer	Commit	Stages	
<div>passed</div>	#10909 latest		release 4a643309 Saved modified emacs b...	<div>✓✓</div>	<div>🕒 00:01:02</div> <div>📅 3 days ago</div> <div></div>
<div>passed</div>	#10908 latest		master 4a643309 Saved modified emacs b...	<div>✓✓</div>	<div>🕒 00:01:02</div> <div>📅 3 days ago</div> <div></div>
<div>passed</div>	#10907		master 4e1301f7 fixed folder name in root ...	<div>✓✓</div>	<div>🕒 00:01:05</div> <div>📅 3 days ago</div> <div></div>
<div>passed</div>	#8363		release a5954f38 Push deadline	<div>✓✓</div>	<div>🕒 00:00:57</div> <div>📅 2 weeks ago</div> <div></div>
<div>passed</div>	#8362		release bd544248	<div>✓✓</div>	<div>🕒 00:00:56</div> <div></div> <div></div>

Kari Systä
@systa

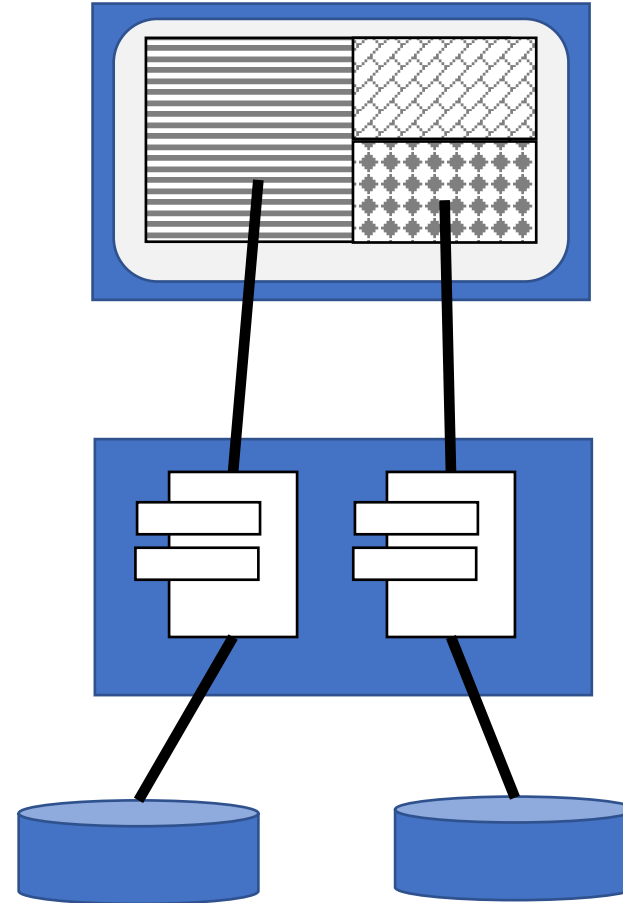
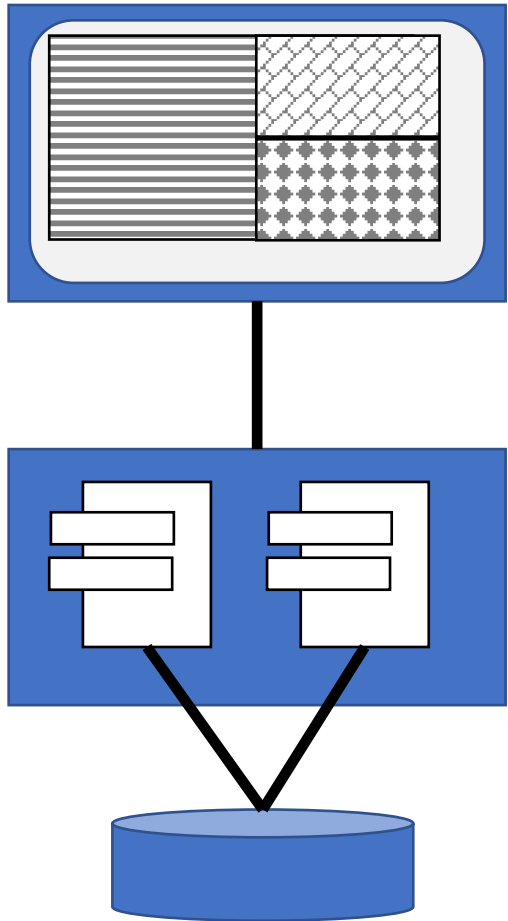
Set status

Profile

Settings

Sign out

Build around business capabilities?



Projects ▾Groups ▾ActivityMilestonesSnippets

Search or jump to...

19




















0

0

Faculty of Information Technology and Communication Sciences > ... > TIE-23536 > plusa-syksy2019 > Pipelines

All 91 Pending 0 Running 0 Finished 91 BranchesTags

Run Pipeline

Status	Pipeline	Triggerer	Commit	Stages
<div>passed</div>	#10909 latest		 release  4a643309  Saved modified emacs b...	<div><div>✓</div><div>✓</div></div> <div>00:01:02 3 days ago</div>
<div>passed</div>	#10908 latest		 master  4a643309  Saved modified emacs b...	<div><div>✓</div><div>✓</div></div> <div>00:01:02 3 days ago</div>
<div>passed</div>	#10907		 master  4e1301f7  fixed folder name in root ...	<div><div>✓</div><div>✓</div></div> <div>00:01:05 3 days ago</div>
<div>passed</div>	#8363		 release  a5954f38  Push deadline	<div><div>✓</div><div>✓</div></div> <div>00:00:57 2 weeks ago</div>
<div>passed</div>	#8362		 release  bd544248	<div><div>✓</div><div>✓</div></div> <div>00:00:56</div>

Kari Systä
@systa

Set status

Profile

Settings

Sign out

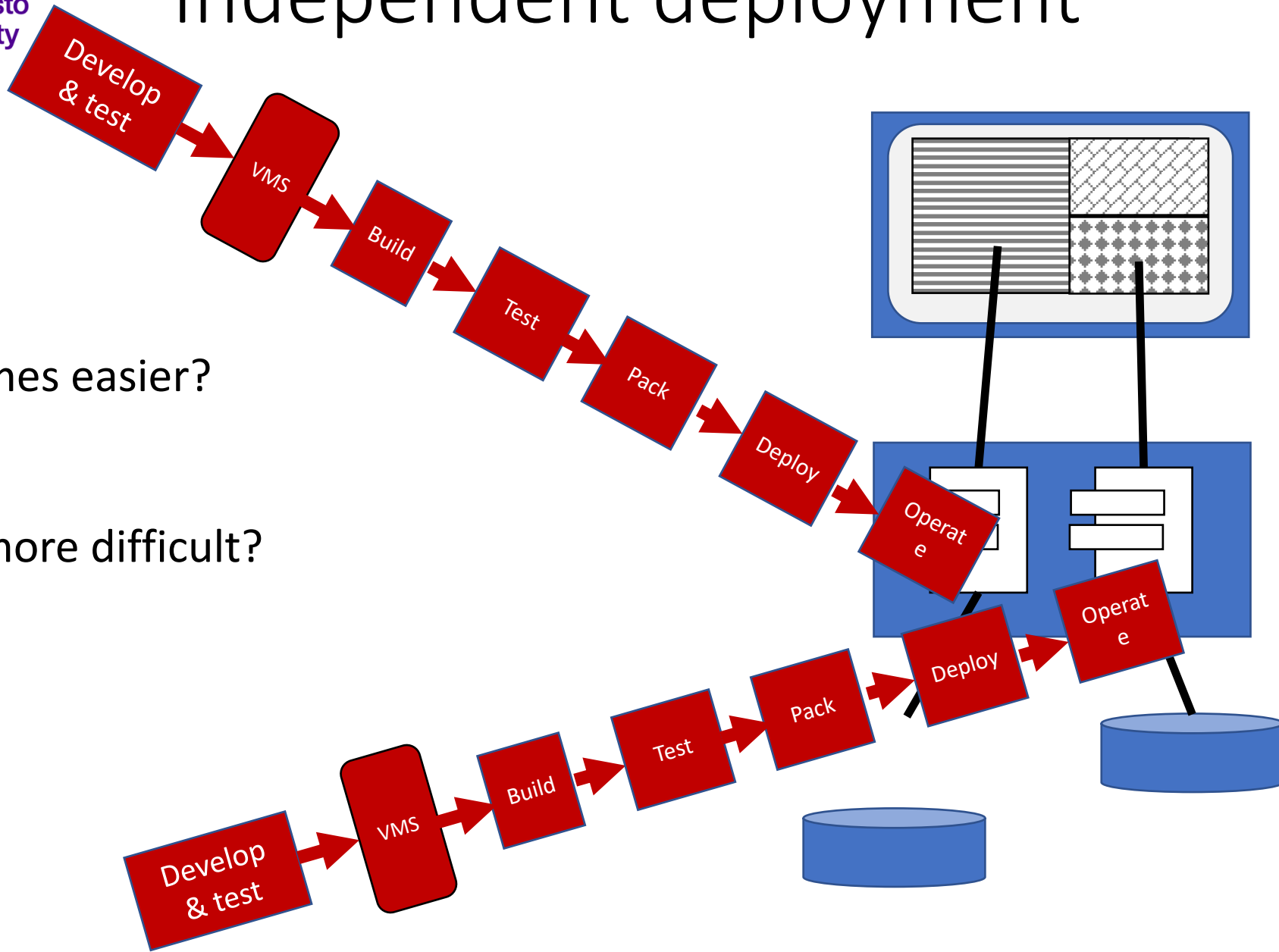
Independent development

- Separate team
 - Two pizza rule
 - Scrum propose 7 ± 2
 - Small team is more efficient
- Independently selected
 - Run-time
 - Libraries
 - Programming language
- Or, arrogant developers want to use their own favorite?

Independent deployment

What becomes easier?

What gets more difficult?



End of 20.10 lecture