=============Micro Services==================

PART 1. Core Concepts in Microservices

1. Cohesion
2. Coupling
3. Immutability in Microservices
4. Open/Close Principle
5. DRY (Don’t Repeat Yourself)
6. SOLID
7. Single Responsibility Principle
8. 8 Fallacies of Distributed Computing
9. Continuous Integration (CI)
10. CAP Theorem
11. 12 Factor App
12. Typical Git workflow for a real project.

PART 2. Introduction to Microservices.

1. Characteristics of a microservices architecture

While there is no precise definition of this **architectural** style, there are certain common **characteristics** around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

1. Benefits of using Microservices Architecture

**Advantages** of **microservices**

Eliminate vendor or technology lock-in: **Microservices** provide the flexibility to try out a new technology stack on an individual service as needed. There won't be as many dependency concerns and rolling back changes becomes much easier. **With** less code in play, there is more flexibility

### Advantages of microservices

The advantages of microservices seem strong enough to have convinced some big enterprise players such as Amazon, Netflix, and eBay to adopt the methodology. Compared to more monolithic design structures, microservices offer:

* **Improved fault isolation**: Larger applications can remain mostly unaffected by the failure of a single module.
* **Eliminate vendor or technology lock-in**: Microservices provide the flexibility to try out a new technology stack on an individual service as needed. There won’t be as many dependency concerns and rolling back changes becomes much easier. With less code in play, there is more flexibility.
* **Ease of understanding:**With added simplicity, developers can better understand the functionality of a service.
* **Smaller and faster deployments**: Smaller codebases and scope = quicker deployments, which also allow you to start to explore the benefits of Continuous Deployment.
* **Scalability**: Since your services are separate, you can more easily scale the most needed ones at the appropriate times, as opposed to the whole application. When done correctly, this can impact cost savings.

### Disadvantages of microservices

Microservices may be a hot trend, but the architecture does have drawbacks. In general, the main negative of microservices is the complexity that any distributed system has.

Here’s a list of some potential pain areas and other cons associated with microservices designs:

* **Communication between services is complex**: Since everything is now an independent service, you have to carefully handle requests traveling between your modules. In one such scenario, developers may be forced to write extra code to avoid disruption. Over time, complications will arise when remote calls experience latency.
* **More services equals more resources**: Multiple databases and transaction management can be painful.
* **Global testing is difficult**: Testing a microservices-based application can be cumbersome. In a monolithic approach, we would just need to launch our WAR on an application server and ensure its connectivity with the underlying database. With microservices, each dependent service needs to be confirmed before testing can occur.
* **Debugging problems can be harder**: Each service has its own set of logs to go through. Log, logs, and more logs.
* **Deployment challengers**: The product may need coordination among multiple services, which may not be as straightforward as deploying a WAR in a container.
* **Large vs small product companies**: Microservices are great for large companies, but can be slower to implement and too complicated for small companies who need to create and iterate quickly, and don’t want to get bogged down in complex orchestration.

Of course, with the right kind of automation and tools and the properly trained staff, all the above drawbacks can be addressed.

### Deployment of microservices

Now that we understand microservices, how are they deployed?

The best way to deploy microservices-based applications is within containers, which are [complete virtual operating system environments](https://cloudacademy.com/blog/container-virtualization/) that provide processes with isolation and dedicated access to underlying hardware resources. One of the biggest names in container solutions right now is [Docker](https://www.docker.com/" \t "_blank), which you can learn more about in our [Getting Started Course](https://cloudacademy.com/cloud-computing/introduction-to-docker-course/).

Virtual machines from infrastructure providers like Amazon Web Services (AWS) can also work well for microservices deployments, but relatively lightweight microservices packages may not leverage the whole virtual machine, potentially reducing their cost-effectiveness.

Code deployments can also be completed using an Open Service Gateway Initiative (OSGI) bundle. In this use case, all application services will be running under one Java virtual machine, but this method comes with a management and isolation tradeoff.

### How to move forward with microservices

As application development trends continue to evolve, the debate between using microservices or leveraging traditional monolithic architectures will only become more pronounced. In the end, developers must do their due diligence and understand what works for their specific use cases.

For smaller companies, starting with a monolithic application can be simpler, faster, and cheaper — and if the product hasn’t gotten too mature, it can still be migrated to microservices at an appropriate time. The huge companies with millions of users are obvious examples of the best use case for microservices, as they need to ensure the uptime, scalability that the added modularity can provide.

### A quick video review of the advantage of microservices

Check out our video below where Cloud Academy DevOps lead Jeremy Cook compares monolithic vs microservice architectures. And you can get further info straight from the source: our course on [.Net Microservices – Refactors and Design.](https://cloudacademy.com/course/net-microservices-refactor-design-course-1/c2l1-introduction/)

[h <https://youtu.be/h5dYTRpXopY>ttps://yout <https://youtu.be/h5dYTRpXopY>u.be/h5dYTRpXopY](https://youtu.be/h5dYTRpXopY)

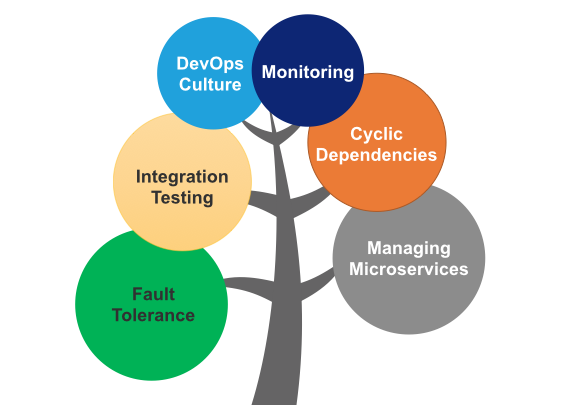
1. Challenges in Microservices.
2. **22.54k Views**

Join the DZone community and get the full member experience.

[**JOIN FOR FREE**](https://dzone.com/articles/challenges-in-implementing-microservices)

We live in a world of microservices. "Monolith to microservices" is a phrase we hear from more than 70% of technology leaders today. The benefits are well-documented: increased resilience, improved scalability, faster time to market. Like most transformational trends, implementing microservices poses its own challenges. It is imperative that these challenges are well understood, or be prepared for your project to never see the light of day, or finish it to only see that many of the foreseen benefits are not being achieved.

Here are some of the top challenges that organizations face in their microservices journey:



**Managing Microservices**

As the number of microservices increases, managing them gets more challenging. It is important that management is planned before or while microservices are being built. While the modularity helps, things can very quickly get out of hand if not managed well. Many engineering leaders have stated that the mismanagement of these services is as much a problem as problems faced during the initial stages of the transformation from monolithic applications.

Developing microservices management tooling on your own, although a valid option, can be complex and cumbersome. We recommend looking to acquire a platform whose capabilities include microservices management.

**Monitoring**

The traditional forms of monitoring and diagnostics will not align well with microservices since you have multiple services making up the same functionality previously supported by a single application. When a problem arises in the application, finding the root cause can be challenging if you do not have a means of monitoring and tracking the path a specific request took, like how many and which microservices were traversed for a specific request coming from a user interface.

We have seen customers who struggle to analyze the chain of communication across these services and where issues were potentially introduced. [This](https://www.youtube.com/watch?v=smEuX-Hq6RI) video is a good watch on this topic.

**Embracing DevOps Culture**

Separate teams need agility, autonomy, and continuous delivery to be able to deliver initial releases and subsequent iterative changes. A lack of DevOps culture can bottle up releases and impact the overall time to market and the response to business requests and issues.

**Fault Tolerance**

It is important that individual services do not bring down the overall system. Fault tolerance at the service level, and more importantly, at the overall solution level, is critical. Given the complexity of a microservices environment and the complex dependency chains, failure is inevitable. Microservices need to be able to withstand both internal and external failures. Robust resiliency testing is key to successful issue preparedness.

**Testing**

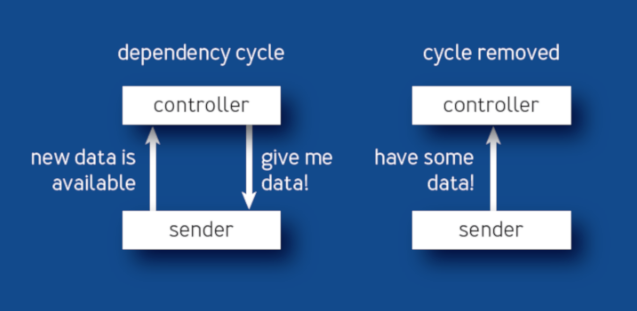
Testing is much more complex in a microservices environment due to the different services, their integration, and interdependencies. The team members responsible for quality assurance need to be knowledgeable on the order and channels of communications between services to have full coverage in their test cases. The asynchronous aspect of microservices also makes it harder to test in lower environments. Indistinct behaviors from microservices are harder to predict and validate.

More details on microservice testing challenges can be found [here](http://www.alohatechnology.com/blog/testing-challenges-in-a-microservices-environment.html) and in [this insightful post](https://bit.ly/2meWzaF).

**Design With Failure in Mind**

While this is counter-intuitive to many, expecting failure scenarios and building a robust set of microservices is imperative to a successful implementation. When more failure situations are predicted during design, the more exception handling mechanisms will be built and seamless resolution of issues will be handled better. This is easier said than done.

**Cyclic Dependencies**

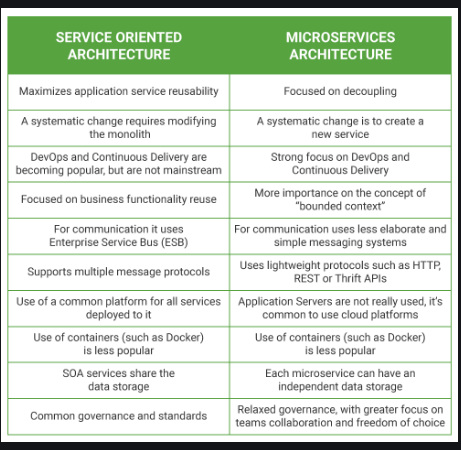


*Source:*[*queue.acm.org*](http://queue.acm.org/)*.*

Dependency management across different services and their functionality is very important and cyclic dependencies can be a headache if not identified and resolved promptly. In microservice architecture, you’re even more vulnerable to errors coming from dependency issues. Decisions made around upgrades on related services with these dependencies are critical. [This post](https://queue.acm.org/detail.cfm?id=3277541) discusses tracking and controlling dependencies in detail.

Good luck with your microservices journey!

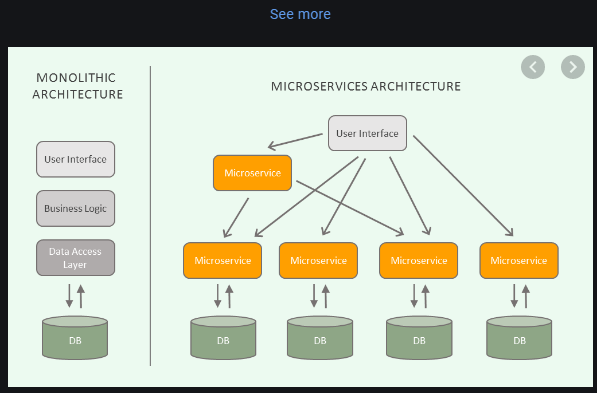
1. Difference between Microservices and SOA



PART 3. Microservices Interview Questions .

1. How will you define Microservices Architecture

**Microservices** - also known as the **microservice architecture** - is an **architectural** style that structures an application as a collection of services that **are**. Highly maintainable and testable. Loosely coupled. Independently deployable. Organized around business capabilities.

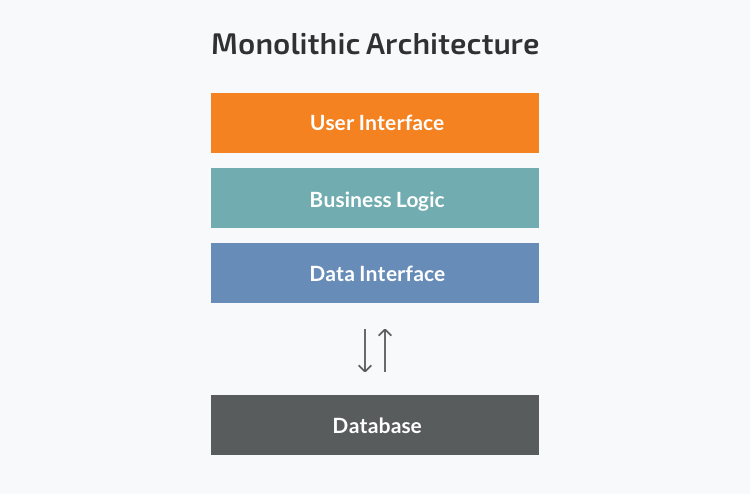


1. What is Domain Driven Design?
2. What is Bounded Context?
3. What is polyglot persistence? Can this idea be used in monolithic applications as well?
4. Why Microservices are better than Monoliths?

## MONOLITHIC ARCHITECTURE

**The monolithic architecture** is considered to be a traditional way of building applications. A monolithic application is built as a single and indivisible unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database. It is unified and all the functions are managed and served in one place.

Normally, monolithic applications have one large code base and lack modularity. If developers want to update or change something, they access the same code base. So, they make changes in the whole stack at once.

[](https://s3-eu-west-1.amazonaws.com/img3.n-ix.com/wp-content/uploads/2018/10/01181553/25-facts-about-outsourced-software-engineering-in-Eastern-Europe-021.jpg)

### Strengths of the Monolithic Architecture

* **Less cross-cutting concerns.** Cross-cutting concerns are the concerns that affect the whole application such as logging, handling, caching, and performance monitoring. In a monolithic application, this area of functionality concerns only one application so it is easier to handle it.
* **Easier debugging and testing.**In contrast to the microservices architecture, monolithic applications are much easier to debug and test. Since a monolithic app is a single indivisible unit, you can run end-to-end testing much faster.
* **Simple to deploy. Another advantage associated with the simplicity of monolithic apps is easier deployment. When it comes to monolithic applications, you do not have to handle many deployments – just one file or directory.**
* **Simple to develop. As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.**

### Weaknesses of the Monolithic Architecture

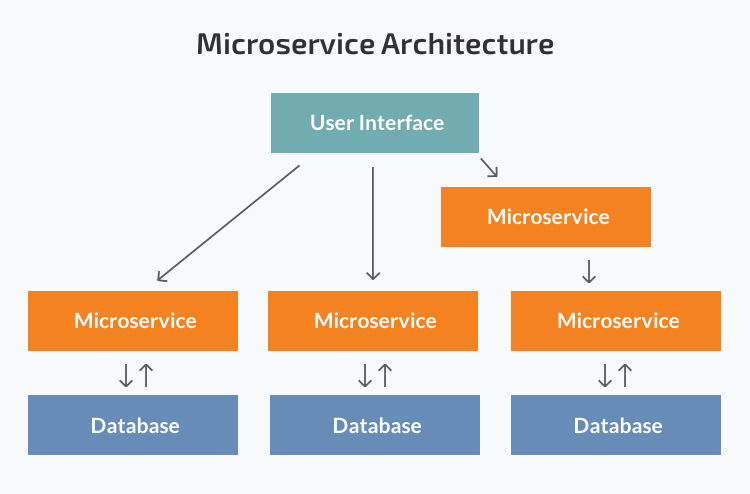
* **Understanding. When a monolithic application scales up, it becomes too complicated to understand. Also, a complex system of code within one application is hard to manage.**
* **Making changes. It is harder to implement changes in such a large and complex application with highly tight coupling. Any code change affects the whole system so it has to be thoroughly coordinated. This makes the overall development process much longer.**
* **Scalability.** You cannot scale components independently, only the whole application.
* **New technology barriers.** It is extremely problematic to apply a new technology in a monolithic application because then the entire application has to be rewritten.

## MICROSERVICES ARCHITECTURE

While a monolithic application is a single unified unit, a **microservices architecture** breaks it down into a collection of smaller independent units. These units carry out every application process as a separate service. So all the services have their own logic and the database as well as perform the specific functions.

In short, 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.Martin Fowler

Within a microservices architecture, the entire functionality is split up into independently deployable modules which communicate with each other through defined methods called APIs (Application Programming Interfaces). Each service covers its own scope and can be updated, deployed, and scaled independently.

[](https://s3-eu-west-1.amazonaws.com/img3.n-ix.com/wp-content/uploads/2018/10/01181617/25-facts-about-outsourced-software-engineering-in-Eastern-Europe-03.jpg)

### Strengths of the Microservice Architecture

* **Independent components.**Firstly, all the services can be deployed and updated independently, which gives more flexibility. Secondly, a bug in one microservice has an impact only on a particular service and does not influence the entire application. Also, it is much easier to add new features to a microservice application than a monolithic one.
* **Easier understanding.**Split up into smaller and simpler components, a microservice application is easier to understand and manage. You just concentrate on a specific service that is related to a business goal you have.
* **Better scalability.**Another advantage of the microservices approach is that each element can be scaled independently. So the entire process is more cost- and time-effective than with monoliths when the whole application has to be scaled even if there is no need in it. In addition, every monolith has limits in terms of scalability, so the more users you acquire, the more problems you have with your monolith. Therefore, many companies, end up rebuilding their monolithic architectures.

For instance, our partner Currencycloud needed to migrate to microservices due to the growing number of transactions their platform processed. Founded in 2012, the company offers a global B2B payments platfrom. Initially, their monolithic architecture could handle the number of transactions they had. Yet with the company’s growing success, they needed a more efficient solution they could scale even further in the future. So they have switched to microservices.

* **Flexibility in choosing the technology.** The engineering teams are not limited by the technology chosen from the start. They are free to apply various technologies and frameworks for each microservice.
* **The higher level of agility.**Any fault in a microservices application affects only a particular service and not the whole solution. So all the changes and experiments are implemented with lower risks and fewer errors.

### Weaknesses of the Microservice Architecture

* **Extra complexity.** Since a microservices architecture is a distributed system, you have to choose and set up the connections between all the modules and databases. Also, as long as such an application includes independent services, all of them have to be deployed independently.
* **System distribution.**A microservices architecture is a complex system of multiple modules and databases so all the connections have to be handled carefully.
* **Cross-cutting concerns.**When creating a microservices application, you will have to deal with a number of cross-cutting concerns. They include externalized configuration, logging, metrics, health checks, and others.
* **Testing.**A multitude of independently deployable components makes testing a microservices-based solution much harder.

## SO WHICH SOFTWARE ARCHITECTURE SUITS YOUR SOLUTION AND YOUR BUSINESS BEST?

### Choosing a monolithic architecture

* **Small team.**If you are a startup and your team is small, you may not need to deal with the complexity of the microservices architecture. A monolith can meet all your business needs so there is no emergency to follow the hype and start with microservices.
* **A simple application.**Small applications which do not demand much business logic, superior scalability, and flexibility work better with monolithic architectures.
* **No microservices expertise. Microservices require profound expertise to work well and bring business value. If you want to start a microservices application from scratch with no technical expertise in it, most probably, it will not pay off.**
* **Quick launch. If you want to develop your application and launch it as soon as possible, a monolithic model is the best choice. It works well when you aim to spend less initially and validate your business idea.**

### Choosing a microservices architecture

* **Microservices expertise. Without proper skills and knowledge, building a microservice application is extremely risky. Still, just having the architecture knowledge is not enough. You need to have DevOps and Containers experts since the concepts are tightly coupled with microservices. Also, domain modelling expertise is a must. Dealing with microservices means splitting the system into separate functionalities and dividing responsibilities.**
* **A complex and scalable application. The microservices architecture will make scaling and adding new capabilities to your application much easier. So if you plan to develop a large application with multiple modules and user journeys, a microservice pattern would be the best way to handle it.**
* **Enough engineering skills.** Since a microservice project comprises multiple teams responsible for multiple services, you need to have enough resources to handle all the processes.

1. Isn’t in process communication in monolithic application faster than tons of remote network calls in microservices architecture?

<https://dzone.com/articles/building-microservices-inter-process-communication-2>

1. How microservices are different than SOA?
2. What is difference between small-services and microservices?
3. What are benefits of using microservices architecture?
4. How to partition a large application into microservices architecture, correctly?
5. How big a single microservice should be?
6. How do microservices communicate with each other?
7. What shall be preferred communication style in microservices: synchronous or asynchronous?
8. What is difference between Orchestration and Choreography in microservices context?
9. How to maintain ACID in microservice architecture?
10. How frequent a microservice be released into production?
11. How to achieve zero-downtime during the deployments?
12. How to achieve zero downtime deployment(blue/green) when there is a database change?
13. How to slowly move users from older version of application to newer version?
14. How will you monitor fleet of microservices in production?
15. How will you troubleshoot a failed API request that is spread across multiple services?
16. What are different layers of a single microservice?
17. How will you develop microservices using Java?
18. Is it a good practice to deploy multiple microservices in a single tomcat container (servlet container)?
19. What are Cloud Native applications?
20. What is Spring Boot?
21. What is Spring Cloud?
22. What is difference between application.yml and bootstrap.yml?
23. How will you implement service discovery in microservices architecture?
24. How does Eureka Server work?
25. How to externalize configuration in a distributed system?
26. How will you use config-server for your development, stage and production environment?
27. What is difference between config first bootstrap and discovery first bootstrap in context of Spring Cloud Config client?
28. How to halt a Spring Boot based microservice at startup if it can not connect to Config Server during bootstrap?
29. How to refresh configuration changes on the fly in Spring Cloud environment?
30. How to achieve client side load balancing in Spring Microservices using Spring Cloud
31. How to use client side load-balancer Ribbon in your microservices architecture?
32. How to use both LoadBalanced as well as normal RestTemplate object in the single microservice?
33. How will you make use of Eureka for service discovery in Ribbon Load Balancer?
34. Can we use Ribbon without eureka?
35. How will you use ribbon load balancer programmatically?
36. What is difference between @EnableEurekaClient and @EnableDiscoveryClient?
37. How to make microservices zone aware so as to prefer same zone services for inter-service communication using Spring Cloud?
38. How to list all instances of a single microservice in Spring Cloud environment
39. What is API Gateway?
40. How to protect internal endpoints leaking from API Gateway?
41. How to protect Sensitive Security Tokens from leaking into downstream system?
42. How to retry failed requests at some other available instance using Client Side Load Balancer?
43. What is Circuit Breaker Pattern?
44. What are Open, Closed and Half-Open states of Circuit Breaker
45. What are use-cases for Circuit Breaker Pattern?
46. What are benefits of using Circuit Breaker Pattern?
47. Can circuit breaker be used in asynchronous communication?
48. What is Hystrix?
49. What are main features of Hystrix library?
50. How to use Hystrix for fallback execution?
51. When not to use Hystrix fallback on a particular microservice?
52. How will you ignore certain exceptions in Hystrix fallback execution?
53. What is Strangulation Pattern in microservices architecture?
54. What is Circuit Breaker?
55. What is difference between using a Circuit Breaker and a naive approach where we try/catch a remote method call and protect for failures?
56. What is Request Collapsing feature in Hystrix?
57. What is difference between Circuit Breaker and Hystrix?
58. Where exactly should I use Circuit Breaker Pattern?
59. What is bulkhead design pattern?
60. How does Hystrix implements Bulkhead Design Pattern?
61. What is Hystrix approach to Bulkhead Pattern?
62. In microservices architecture, what are smart endpoints and dumb pipes?
63. What is difference between Semaphore and ThreadPool based configuration in Hystrix?
64. How to handle versioning of microservices?
65. What is difference between partitioning microservices based on technical capabilities vs business capabilities? Which one is better?
66. Running Spring boot app at different port on server startup.
67. How will you run certain business logic at the app startup?
68. How to correctly implement a reporting microservice in a distributed system?
69. What is Event Sourcing and CQRS? When should it be used? Should be use it for the entire 84 system?
70. How to send business errors from a RESTful microservice to client application?
71. Is it a good idea to share common database across multiple microservices?
72. How will you make sure that the email is only sent if the database transaction does not fail?
73. How will you atomically update the database and publish an event to message broker from single transaction?
74. How will you propagate security context of user when one microservice calls another microservice on behalf of user?

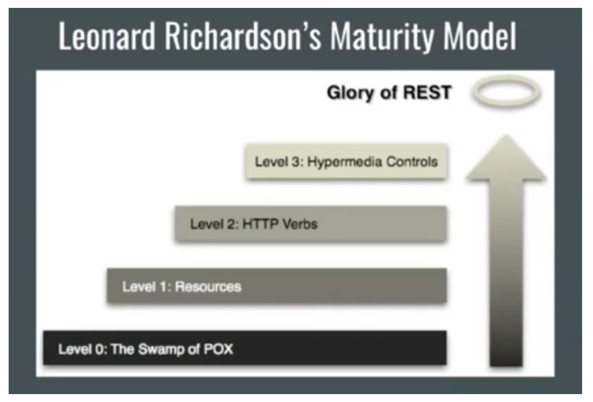
<https://nordicapis.com/how-to-control-user-identity-within-microservices/>

1. What is Token Relay in Spring Security?
2. How to Enable Token Relay?
3. How to revoke Access and Refresh Tokens on data breach to limit the damage?
4. Shall Authentication and Authorization be one service?

**Difference between Authentication and Authorization**. ... **Authentication** means confirming your own identity, while **authorization** means granting access to the system. In simple terms, **authentication** is the process of verifying who you are, while **authorization** is the process of verifying what you have access to

1. What is API Key security?
2. What are best practices for microservices architecture?

## ****Best Practice #1**** — Try to Reach the Glory of REST



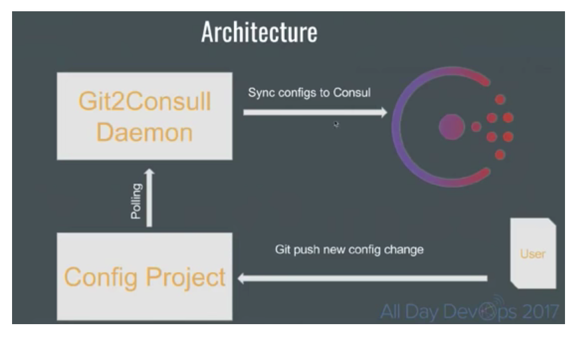
Realize the benefits (nay, the glory) of REST APIs. Looking at Leonard Richardson’s Maturity Model, there are four levels of using REST. You can start a level 0, which is soft resources, using one endpoint; then Level 1, which has different resources, but has the same HTTP method; Level 2, which uses different HTTP methods, such as POST, PUT, DELETE, etc.; finally, Level 3 - you have navigational resources on your API responses. Behold, the glory!

## ****Best Practice #2**** — Use Spring HATEOAS

This helps you use navigable, restful APIs.

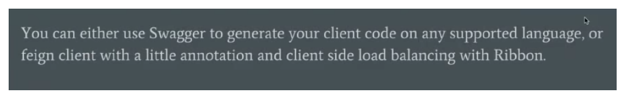
## ****Best Practice #3**** — Use Distributed Configuration

This way, you aren’t configuring 50 different configurations one at a time. Consul can be used to keep config at Key/Value.



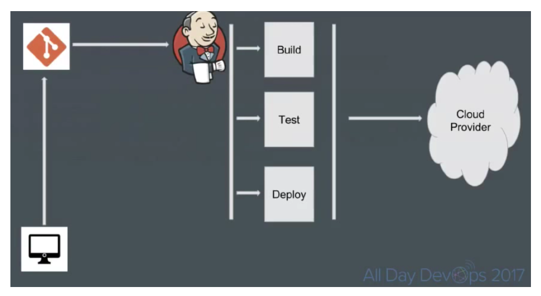
## ****Best Practice #4 —****Client Code Generation

Hüseyin suggests, “either using Swagger to generate your client code on any supported language or use feign client with a little annotation and client-side load balancing with Ribbon.”



## ****Best Practice #5 —****Continuous Delivery

Hüseyin walks through some examples using [Jenkins](https://jenkins.io/) and [Docker](https://www.docker.com/" \t "_blank).



## ****Best Practice #6**** — Monitor

In fact, monitor everything.



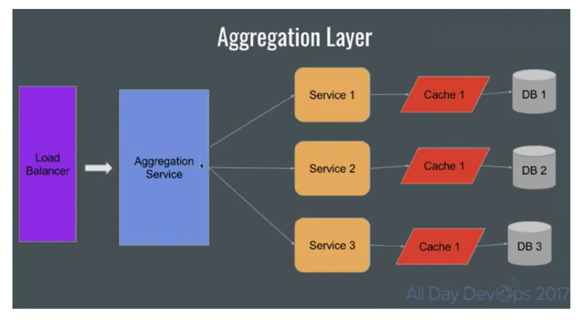
## ****Best Practice #7**** — Logging

Hüseyin points out there are numerous Docker log drivers. He walks through examples with GELF (Graylog Extended Log Format).

## ****Best Practice #8 —**** APM

Application performance management. This collects extra details to help you troubleshoot issues. [Zipkin](https://zipkin.io/" \t "_blank) is an open source option Hüseyin walks through.

## ****Best Practice #9**** — API Gateways to Aggregate Data to Specific Clients



## ****Best Practice #10**** — Event Sourcing and CQRS (Command and Query Responsibility Segregation)

A Command alters the state of an object, but does not return data. A Query returns data, but does not alter the state of the object.

You can watch Hüseyin’s full talk [here](https://youtu.be/CTnMUE06oZI). He walks through technical examples and solutions to each best practice.

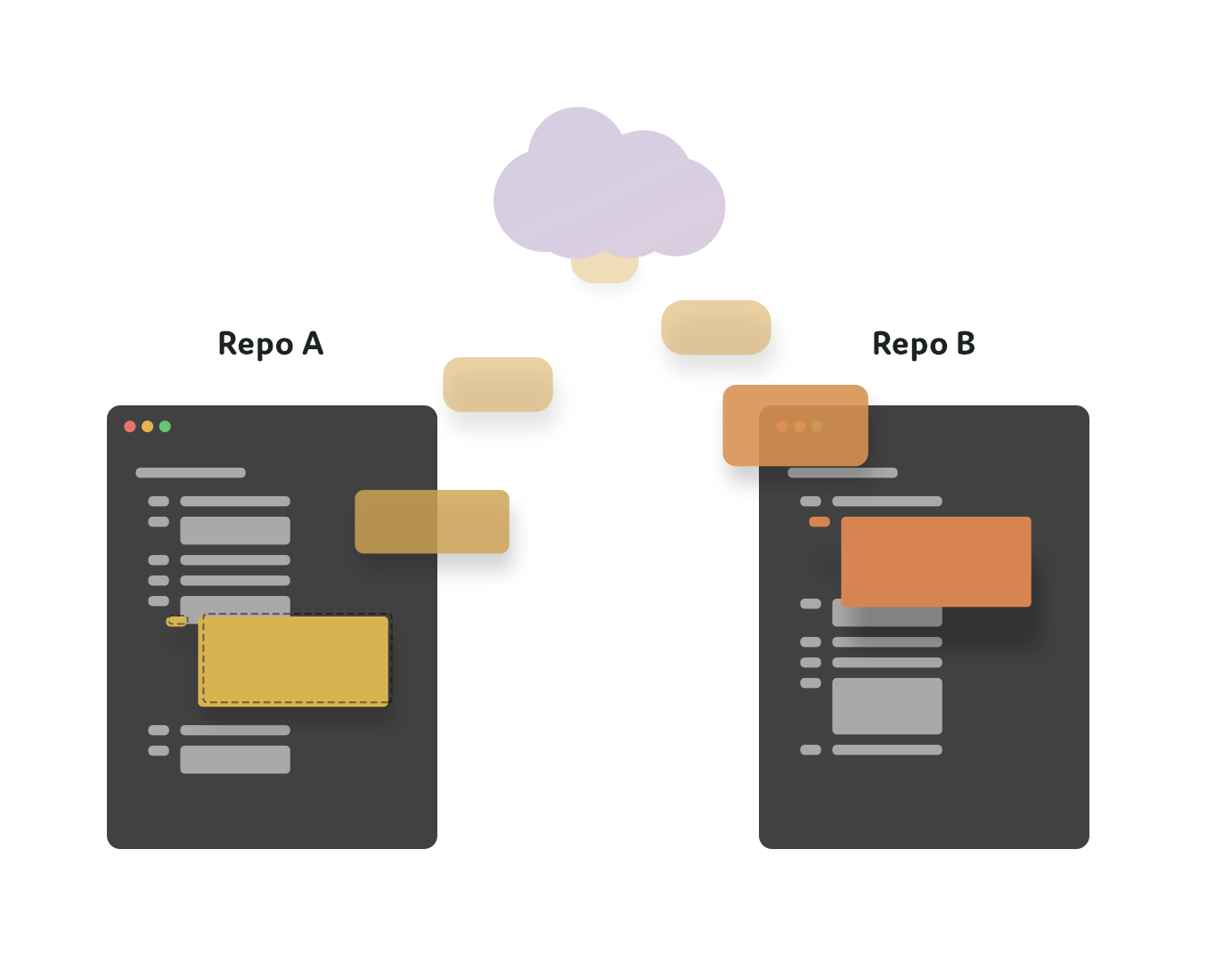
Craving more on Modern Infrastructure and DevOps? Binge watch any of the 20 sessions, free of charge, from All Day DevOps [here](https://www.alldaydevops.com/all-day-devops-2017-recordings?__hstc=31049440.bebf30854e005c7231b9634c07d82b9f.1529594894579.1529594894579.1529602173423.2&__hssc=31049440.1.1529602173423&__hsfp=2652442231).

1. Shall we share common domain models or DTOs across microservices?
2. How to share common code across multiple microservices?

A microservice architecture is great for building scalable codebases with less coupling, better separation of concerns, improved resilience, combining different technologies, and, most of all, better modularity and reusability for the components that build it.

However, modularity and reuse may often result in high-coupling or code duplications. Having different services tied to the same shared-lib undermines why we use services in the first place.

Using new open source technologies [like Bit](https://bit.dev/), it becomes easier and more effective than ever to share and reuse common code between our microservices. Let's see why and how.

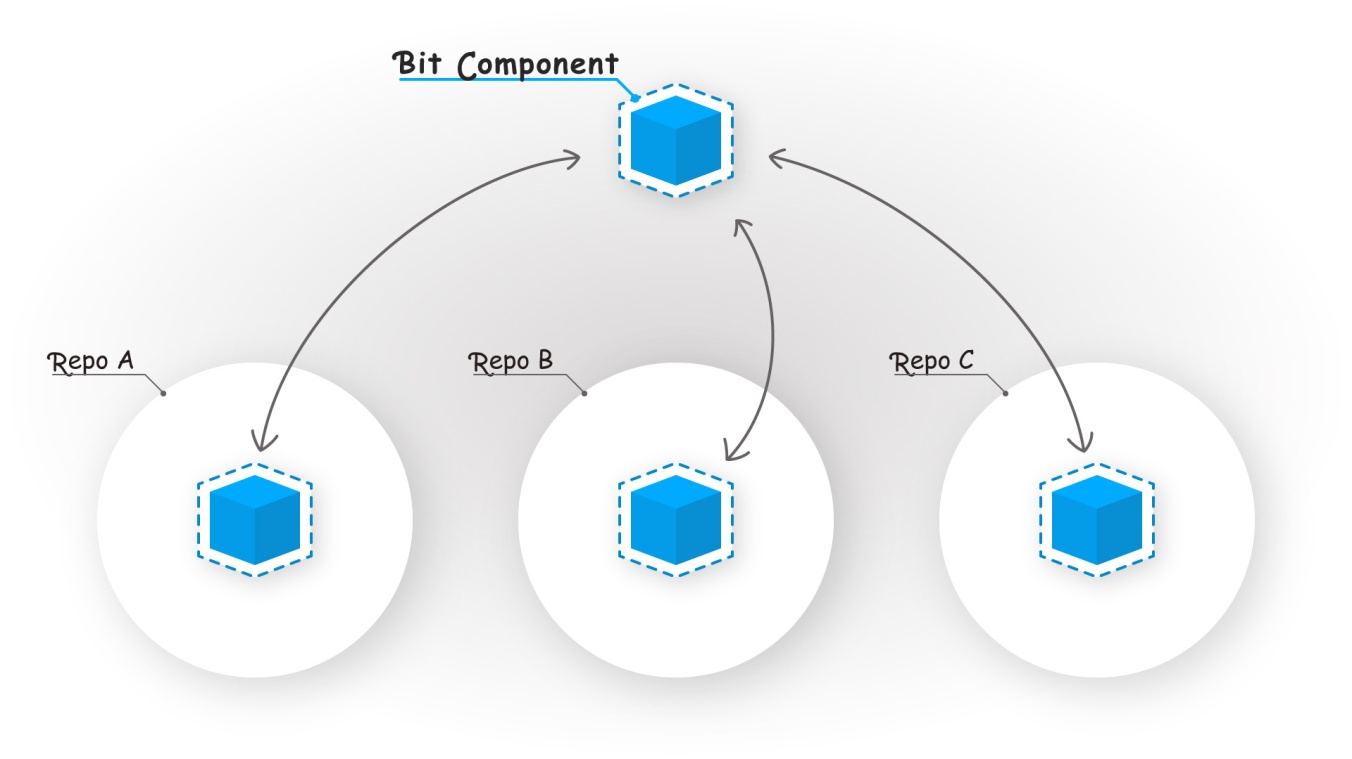
**Sharing Code Between Microservices**

Before explaining how Bit can help solve this problem, let's set the main goals we want to achieve.

1. Sharing common code between our microservices while keeping our code DRY.
2. Avoiding coupling through shared-libs which eliminates the advantages of separating development process.
3. Enable simple changes and sync to the code we share between our microservices.

Microservices are prawned to code duplications. For example, any service which is used by other services will cause all these other services to duplicate the code needed in order to use that service's API.

Creating an npm package (with a new repo) for any such piece of code is highly impractical, and will generate a lot of overhead while making it harder to make changes to the code.

**No Shared Libs, No Coupling  
**

[Bit](https://bitsrc.io/)is an open source project which brings a whole new approach to how we share and reuse code in our microservice architecture. Using Bit, you don't have to create a new repository or configure packages to share code instead of duplicating it.

Instead, you can simply define reusable parts of any existing repository and share to the other repositories — as packages or as tracked source code. This way you can make parts of any service reusable from other services without changing a single line of code in your codebase, creating more repos, or coupling your microservices together.

The best part is, Bit also lets you make changes to the code you share with any other service- so you can develop and modify that code from basically any repository.

**Example Workflow**

Instead of coupling your microservices together via common libraries, you can simply isolate and sync any reusable code using the Bit's ability to isolate and track source code among projects. You can even still install this code with NPM in different repos, and still make changes from any end.

Let’s use Bit to isolate and share the reusable components left-pad, some-logic, and hello-world in the following project’s directory structure.

$ tree

.

├── App.js

├── App.test.js

├── favicon.ico

├── index.js

└── src

└── common

├── left-pad

├── some-logic

└── hello-world

First, let’s [install Bit](https://docs.bitsrc.io/docs/installation.html).

$ npm install bit-bin -g

Let’s [initialize Bit](https://docs.bitsrc.io/docs/initializing-bit.html) for the project.

$ cd project-directory

$ bit init

Let’s [add the components](https://docs.bitsrc.io/docs/isolating-and-tracking-components.html) to be tracked by Bit.

$ bit add src/common/\* # use a glob pattern to track multiple components in the same path or a single path to track a single component.

Let’s add [build](https://docs.bitsrc.io/docs/building-components.html) and [test](https://docs.bitsrc.io/docs/testing-components.html) environments.

Now let’s [lock a version](https://docs.bitsrc.io/docs/versioning-tracked-components.html) and isolate the components from the project.

$ bit tag --all 1.0.0

3 components tagged | 3 added, 0 changed, 0 auto-tagged

Now let’s [share](https://docs.bitsrc.io/docs/organizing-components-in-scopes.html) the components to a remote [Scope](https://bitsrc.io/).

$ bit export username.scopename # Share components to this Scope

exported 3 components to scope username.scopename

Note that using the --eject flag you can also remove an exported component from your source code and add it as a package dependency in your project’s package.json file.

That’s it. You can now [install the components](https://docs.bitsrc.io/docs/installing-components-using-package-managers.html) using your favorite package manager, or use bit import to [bring their source code](https://docs.bitsrc.io/docs/importing-components.html) into any repository, make changes, and sync them across your codebase. So, you eliminate the coupling between services — since you can make changes, update versions, and develop the code from each service — without compraminzing on being able to manage and sync them.

**Conclusion**

Microservices provide increased modularity and separation for your development process. Many services will use the same code, so sharing code between them is critical for your development and maintenance efforts.

However, coupling services through shared libs might ruin the point of having multiple different services. Creating different repos to publish every few code lines as a package to npm isn't practical.

Using new technologies like [Bit](https://bitsrc.io/)we can have the best of both worlds: easily share common code between our microservices, make and sync changes from any end, and avoid the coupling created via adding third-party shared libraries.

1. What is continuous delivery?
2. How will you improve the performance of distributed system?
3. How will you implement caching for microservices?
4. Which protocol is generally used for client to service and inter-service communication?.
5. What are advantages of using asynchronous messaging within microservices architecture?
6. What is good tool for documenting Microservices?
7. How will you integrate Swagger into your microservices?
8. What are common properties for a Spring Boot project

PART 4. Security in Microservices.

1. Why Basic Authentication is not suitable in Microservices Context?
2. Why OAuth2?
3. How OAuth2 Works?
4. What are different OAuth2 Roles?
5. What are different OAuth 2.0 grant types (OAuth flows)?
6. When shall I use resource owner credentials
7. When shall I use Authorization Code grant
8. When shall I use client credentials?
9. OAuth2 and Microservice
10. What is JWT?
11. What are usecases for JWT
12. How does JWT looks like
13. What is AccessToken and RefreshToken
14. How to use a RefreshToken to request a new AccessToken?
15. How to call the protected resource using AccessToken?
16. Can a refreshToken be never expiring? How to make refreshToken life long valid?
17. Generate AccessToken for Client Credentials.
18. Why there is no RefreshToken support in Oauth2 Client Credentials workflow?
19. How to implement the Logout functionality using JWT?
20. Security in inter-service communication
21. How to setup multiple authentications in Spring Security?
22. What is purpose of @EnableResourceServer?
23. What is purpose of @EnableOAuth2Sso?
24. What is purpose of @EnableOAuth2Client?
25. How can we add custom claims to JWT AccessToken?
26. Security Best Practice
27. How to enable spring security at service layer?

PART 5. Testing Spring Boot based Microservices

1. Tools and Libraries available for testing
2. What is Mike Cohn’s Test Pyramid?
3. Testing Strategies
4. Mock vs Stub?
5. Unit Testing
6. Integration Tests
7. Contract-Driven Tests
8. End to End Tests
9. Best Practices in Testing

For all these questions, you can buy my ebook:

Buy my ebook for complete question bank

Most of these questions has been answered in my eBook "Cracking the Core Java Interview" updated on June 2018, that you can buy from this link:

[Buy from Shunya (DRM Free PDF download with updates)](https://books.shunyafoundation.com/?author=munish+chandel)

[Buy from Google Books](https://play.google.com/store/books/author?id=Munish+Chandel)

Top articles in this category:

1. [Top 50 microservices architecture interview questions for Java developers](https://www.javacodemonk.com/top-50-microservices-architecture-interview-questions-for-java-developers-f47631e8)
2. [Spring Cloud and its advantages](https://www.javacodemonk.com/spring-cloud-and-its-advantages-3ac60b2c)
3. [What is Spring Boot and what are its advantages](https://www.javacodemonk.com/what-is-spring-boot-and-what-are-its-advantages-11f4403f)
4. [Is it a good practice to deploy multiple microservices in a single tomcat container](https://www.javacodemonk.com/is-it-a-good-practice-to-deploy-multiple-microservices-in-a-single-tomcat-container-a638a7c6)
5. [How will you Partition a typical e-shop into Microservices Architecture](https://www.javacodemonk.com/how-will-you-partition-a-typical-e-shop-into-microservices-architecture-b1385fed)
6. [Why Microservices are better than Monoliths](https://www.javacodemonk.com/why-microservices-are-better-than-monoliths-1dc2aad2)