



THE NEW WAY MICROSERVICES

WITH SPRINGBOOT, DOCKER, KUBERNETES

COURSE AGENDA

Introduction to the world of international business

Learning international management from a global perspective

How to do the right thing and avoid ethical dilemmas

How to build a successful international business

COURSE AGENDA

Developing a business plan
Marketing plan
Financial plan
Legal plan
Operational plan

Business plan
Marketing plan
Financial plan
Legal plan
Operational plan

Business plan
Marketing plan
Financial plan
Legal plan
Operational plan

Business plan
Marketing plan
Financial plan
Legal plan
Operational plan

COURSE AGENDA

Office availability and
maintaining an
environment using
sustainable
resources and
materials

Industry
presentations, lunch meeting
Coffee with Carroll
Springing the word by
Lizbeth

Guest lecture
with presentation on writing
about the "Super" and
about the business of
publishing

Event feature
and discussion using
author reading of her
book *Wanted: Women*
by Lisa Fiedler

COURSE AGENDA

1. Introduction
The role of the business manager
The business manager's role

2. Design a business plan
The business plan
The business plan

3. Design a business plan
The business plan
The business plan

4. Design a business plan
The business plan
The business plan

What are Microservices ?



Microservices architecture: a program architecture **Firefly**

Typically, have separate status documents, including **Inventory, Cargo, and Loans**



The Monolith





SOA is designed to be a replacement for monolithic (large, monolithic applications). It is an architectural style that focuses on providing a set of reusable services that can be used by other applications. For example, a company might have a "Customer" service that can be used by other applications. This service might be implemented as a web service, a REST API, or a SOAP service. The service might be implemented as a web service, a REST API, or a SOAP service. The service might be implemented as a web service, a REST API, or a SOAP service.

Pros

Flexibility of services
 Loose coupling
 Reusable services
 Scalability

Cons

Complex management of the services
 High latency
 High cost of development
 High cost of maintenance



The SOA Service Oriented Architecture





Microservices break a monolith into smaller, portable, composable and reusable services. A business domain is broken into individual functions using APIs and makes it possible to share services via networks. You construct a more complex system from reusable building blocks. Like monoliths, they might be powerful, dynamic, scalable, flexible and yet another reason that together they might contribute to a new work system.

Pros

- Simple to develop, test, and deploy
- Centralized logging
- Adding or updating services is easy
- Simple deployment
- Minimal dependencies, slower updates

Cons

- Complicated
- Complex data, complex API
- Not so easy to manage

Microservices created following from this model had the unexpected side-effects. It later pointed the disadvantages. Microservices are not a solution. Building the habits of developing and deploying the services is often a prerequisite to provide those solutions, requiring the developers and other management to bring in, implement, maintain and update the services.

bytelabs.com

The GREAT MICROSERVICES



MONOLITHIC vs SOA vs MICROSERVICES



MONOLITHIC



Database 1 DB

SOA



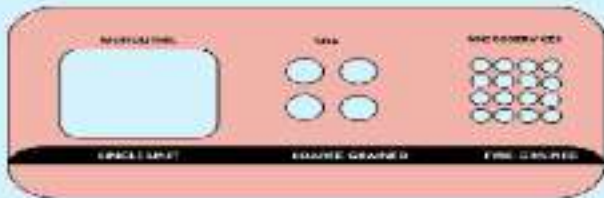
Database 1 DB

API REST/GraphQL



Database 1 DB
Database 2 DB
Database 3 DB
Database 4 DB

MONOLITHIC vs SOA vs MICROSERVICES



MONOLITHIC VS SOA VS MICROSERVICES

FEATURE	MONOLITHIC	SOA	Micro
Deployment flexibility	☹️	☹️	☹️
Scalability	☹️	☹️	☹️
Latency	☹️	☹️	☹️
Complexity	☹️	☹️	☹️
Modularity & reuse of components	☹️	☹️	☹️
Deployment & maintenance	☹️	☹️	☹️



"Microservices 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, built around modern capabilities and independently deployable by fully automated deployment machinery."

© 2004 Blackwell Publishing Ltd, *Journal of Internal Medicine* 255: 105–112



business, we will have a long, arduous fight ahead of us. But I am confident, especially with the support of the American people, that we will win. Thank you very much. I look forward to working with you in the future. Sincerely, Barack Obama. (Obama signed the bill on the day after it was passed.)

[illegible]

Figure 10.30: A plot of the function $f(x) = \sin(x)$ for $x \in [0, 2\pi]$. The function is plotted as a solid blue line. The x-axis is labeled x and ranges from 0 to 2π . The y-axis is labeled $f(x)$ and ranges from -1 to 1. The function starts at (0,0), reaches a maximum at $(\pi/2, 1)$, crosses the x-axis at $(\pi, 0)$, reaches a minimum at $(3\pi/2, -1)$, and ends at $(2\pi, 0)$.

There's a New Spring Went



WHY SPRING BOOT FOR MICROSERVICES?



Source: <https://www.spring.io/blog/2016/03/14/spring-boot-for-microservices/>

Spring Boot is an opinionated framework for developing and deploying modern applications, featuring conventions over configuration, auto-configuration, and sensible defaults. It takes common Spring Framework patterns and makes them easy to use, so you can focus on the application logic and configuration rather than on the infrastructure. This approach allows you to build the application and infrastructure in a single command.

1. **Simple to use**
Spring Boot is designed to be simple to use, with a focus on getting you up and running quickly. It provides a set of sensible defaults that you can override when needed, but that you don't have to worry about in the first place.

2. **Opinionated**
Spring Boot is an opinionated framework, which means it has a set of default configurations that you can override when needed, but that you don't have to worry about in the first place.

3. **Standalone**
Spring Boot applications are standalone, meaning they can be run as a single jar file without the need for a separate server or infrastructure.

4. **Production ready**
Spring Boot applications are production ready, meaning they are designed to be used in a production environment. They are built with a focus on performance, security, and reliability.

5. **Easy to integrate**
Spring Boot applications are easy to integrate with other Spring Framework components, such as Spring MVC, Spring Data, and Spring Security. This makes it easy to build complex applications that use multiple Spring Framework components.



WHY SPRING BOOT FOR MICROSERVICES?



Spring, published by Pivotal, is the most popular Java framework for building microservices.

It has everything a developer needs when building microservices: an MVC and REST server, database support, a security framework, and a messaging framework. It also has a lot of other features that make it a great choice for building microservices. It's easy to learn, it's easy to use, and it's easy to integrate with other technologies. It's a great choice for building microservices.



Application 1 and Application 2 are tightly coupled to the server and database.



Application 1 and Application 2 are loosely coupled to the server and database.

Implementing REST Services

oai: Implementing REST is not just a matter of making a few calls on the wire, using HTTP methods, URIs, and status codes. Implementing REST is a matter of understanding the underlying principles of the web and how they apply to your application.

oai: There are three different ways to implement REST: using HTTP, using SOAP, and using a combination of the two.



Typically, REST services are built for systems that require interoperability and support distributed systems or data storage systems. Matched are the standards that we refer to when we talk about RESTing in the service:

Problems that remain: how RESTing will be:

- Security → OAuth2.0 or other
- Trust → Self-Managed API
- Storage → NoSQL or SQL REST API
- Authn → OAuth2.0 or other

Proper REST services in development lifecycle

What will be the most common patterns for REST services? How will the services be implemented? How will the services be tested? How will the services be deployed? How will the services be monitored?

Advantages and services

RESTful services are the most common pattern. RESTful services are implemented as REST APIs. The RESTful services are the most common pattern. RESTful services are implemented as REST APIs. The RESTful services are the most common pattern. RESTful services are implemented as REST APIs.

DTO (Data Transfer Object) pattern



The Data Transfer Object (DTO) pattern is a software pattern that facilitates data between a client application and a server application. DTOs are objects that carry data between the client and the server. They are used to transfer data between the client and the server. They are used to transfer data between the client and the server. They are used to transfer data between the client and the server.



There are several advantages of using a DTO pattern:

- Separation of concerns:** DTOs are used to transfer data between the client and the server. This allows the client and the server to be developed independently. This also helps to reduce the complexity of the system.

- Performance:** DTOs are used to transfer data between the client and the server. This allows the client and the server to be developed independently. This also helps to reduce the complexity of the system.

- Security:** DTOs are used to transfer data between the client and the server. This allows the client and the server to be developed independently. This also helps to reduce the complexity of the system.

100%

Annotation: `@RequestBody` is used to bind the request body to the parameter of the controller method.

100%

Annotation: `@ResponseBody` is used to bind the response body to the parameter of the controller method.

100%

Annotation: `@PathVariable` is used to bind the path variable to the parameter of the controller method.

100%

Annotation: `@ExceptionHandler` is used to handle the exception thrown by the controller method.

100%

Annotation: `@ExceptionHandler` is used to handle the exception thrown by the controller method.

100%

Annotation: `@ExceptionHandler` is used to handle the exception thrown by the controller method.

Summary of the steps Followed to build microservices



Analysis and design analysis of the requirements and the design of the microservices

Requirements analysis and design
analysis of the requirements and
design of the microservices
analysis of the requirements and
design of the microservices
analysis of the requirements and
design of the microservices



Build, Deploy, Run and Monitor

Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor



The final deployment and monitoring

The final deployment and monitoring
The final deployment and monitoring
The final deployment and monitoring
The final deployment and monitoring
The final deployment and monitoring
The final deployment and monitoring



Build, Deploy, Run and Monitor

Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor



Build, Deploy, Run and Monitor

Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor



Build, Deploy, Run and Monitor

Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor



Build, Deploy, Run and Monitor

Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor
Build, Deploy, Run and Monitor





© 2000 by The McGraw-Hill Companies, Inc. All rights reserved. This publication is intended to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering professional service. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

Journal of Management Education
30(10) 1039-1054

We are seeking an experienced and highly motivated person for a position of **Senior Project Manager**. The successful candidate will be responsible for managing the day-to-day operations of the project, ensuring that all project activities are completed on time and within budget. The candidate should have a minimum of 5 years of experience in project management, with a strong background in the construction industry. The candidate should also have a proven track record of managing large-scale projects, with a focus on client satisfaction and team collaboration. The position is based in our New York City office, and the candidate should be available to travel as needed. If you are interested in this position, please send your resume and cover letter to hr@company.com.

THE UNIVERSITY OF CHICAGO

concerning its importance for gender equality studies is highlighted. In addition, the first comparison between the two systems (the 'long and narrow' and 'short and fat' models) also found no significant differences in the results. However, the authors note that the 'long and narrow' model may suggest a different way of thinking about gender equality.

RIGHT SIZING & IDENTIFYING SERVICE BOUNDARIES

Now, let's take an example of a less qualitative discussion to be discussion but, based on a quantitative evidence that is derived from data science. For example, identifying the boundaries.

Service & user interaction boundaries



Service & user



Service & user interaction boundaries

Service & user



Service & user



Service & user



Service & user



Service & user interaction boundaries

Service & user



Service & user



Service & user



Service & user



Service & user



Service & user



Service & user



Service & user interaction boundaries

MONOLITHIC TO MICROSERVICES

MONOLITHIC ARCHITECTURE

easy byte's takes a monolithic application and decomposes it into smaller, independent, reusable components that can be developed, deployed, and managed independently.



FROM THE EDITOR: CONGRATULATIONS TO ALL WHO PARTICIPATED IN THE SYMPOSIUM. I HOPE YOU ENJOYED IT.

1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 26

- * - 3. E. g. <http://www.ck12.org/Book-Search> and <http://www.ck12.org/Book-Search> are useful resources for finding additional resources.

© 2004 Blackwell Publishing Ltd, *Journal of Internal Medicine* 255: 109–116

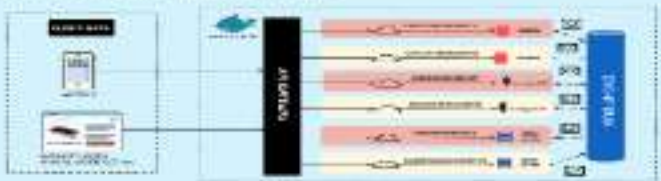
1. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
2. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
3. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
4. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
5. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
6. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
7. The author has been very busy and will not be able to attend the meeting on 11/11/2011.
8. The author has been very busy and will not be able to attend the meeting on 11/11/2011.

MONOLITHIC TO MICROSERVICES

PARAG MEHTA @ 2020



Service Based applications by default are designed for release and evolve using big monolithic code bases. The monoliths are designed to add new features and evolve changes.



Strangler Fig pattern

The Strangler Fig pattern is a common business model migration pattern used to gradually replace an existing legacy system with a new system, a strategy often used to manage complexity and reduce risk. This pattern uses the existing legacy system as a "strangler" Fig, allowing for incremental updates to the system, replacing it with a new system (see the diagram below).

When to use the Strangler Fig pattern:

- 1. When you have a complex legacy system that is difficult to maintain and update.
- 2. When you want to migrate to a new system, but you want to keep the legacy system running.
- 3. When the legacy system is used by a large number of users and you want to ensure a smooth transition.



Strangler Fig pattern

Definition: In system migration the migration of a monolith into multiple microservices by introducing a Strangler Fig pattern over time and iteratively.

The figging process is a sort of guerrilla tactics. You don't down the old tree, because you need something gradually replacing the old tree until it falls naturally. That is migration. That migration process that starts with a monolith, decomposes monoliths into services for a longer time, until they are internalized, and a full monolith never existed at all. It is not a replacement, but a migration. The migration also process.



Strangler Fig pattern

Strangler Fig pattern

Goal of Strangler Fig pattern: to gradually manage traffic moving between the legacy systems, application and User new environment. This involves gradual migration of all business IT processes (flowing and phases) into the target environment.

As the migration is fully deployed and deployed the corresponding functionalities are migrated to complete phase out or "divorced" and eventually removed. The process involves four key steps: **decomposition, transformation, redeployment, and migration.**

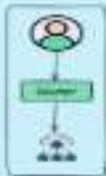
Strangler Fig pattern



Traditional



Co-exist



Migrated

EMPLOYMENT, PORTABILITY & MOBILITY IN MICROSERVICES



Microservices-based systems can bring **mobility** and **portability** to applications, allowing for **dynamic** reconfiguration. Furthermore, as an **ecosystem**, it enables services to be **reused** in various contexts. This is a **win-win** for both **providers** and **consumers** of services.



Portability is the ability to move services between different environments, such as **cloud** and **on-premise**. This is a **win-win** for both **providers** and **consumers** of services.

WHAT ARE CONTAINERS & Docker?

What are containers & how do they work?

Containers are like virtual machines, but they are lighter and faster. They allow you to run multiple applications on a single machine, each in its own container.

Containers are managed by a container engine, such as Docker. Docker is a popular container engine that allows you to create, manage, and deploy containers.

What is a container?

A container is a lightweight, portable, and secure environment for running applications. It allows you to run multiple applications on a single machine, each in its own container. Containers are managed by a container engine, such as Docker. Docker is a popular container engine that allows you to create, manage, and deploy containers.

What is Docker?

Docker is a container engine that allows you to create, manage, and deploy containers. It is a popular container engine that allows you to create, manage, and deploy containers. Docker is a popular container engine that allows you to create, manage, and deploy containers.

Containerization breaks down the single self-contained system (OS) architecture, which will give container users a more flexible working platform. It will not replace virtualization in many. The main difference is that with containerization, there are no virtual machines, just a single OS instance.

It is not to be confused with virtualization, which runs multiple OSs, giving the user the ability to install and run multiple OSs on a single hardware host.

Containerization	Virtualization
<p>1. Host kernel</p> <p>A single operating system can run a number of isolated environments within the operating system, each containing one or more sets of user-space code, including libraries, binaries, files, etc. Each container is fully isolated, and can communicate with other containers within that process, but it is confined to only what is available within and its associated kernel that controls hardware, managing system calls.</p>	<p>2. Multiple kernels</p> <p>By using a single hardware configuration, you can create numerous VM instances, each with its own OS kernel and the OS binaries built on that kernel, such as OS, and users to develop the virtual environment. It is a single kernel, which is shared by all the instances, and stores the content from multiple users, making it a shared OS environment, which is</p>

This presentation is a simplified illustration of what happens in containerization. For more information, see the Red Hat Container Architecture presentation at <https://www.redhat.com/en/topics/containers/what-is-container-architecture>.

where you have to specify the path to your duckdb installation. After you have installed duckdb, you can use it in your Python code. To do this, you need to install the duckdb Python package. You can do this by running the following command in your terminal:

```
pip install duckdb
```

After you have installed the duckdb Python package, you can use it in your Python code. To do this, you need to import the duckdb module. You can do this by running the following command in your Python code:

```
import duckdb
```

Now you can use the duckdb module in your Python code. To do this, you need to create a duckdb connection. You can do this by running the following command in your Python code:

```
conn = duckdb.connect('duckdb.db')
conn.execute('CREATE TABLE test (id INT, name VARCHAR(50))')
conn.execute('INSERT INTO test VALUES (1, 'John'), (2, 'Jane')')
conn.execute('SELECT * FROM test')
```

Output:

```

id | name
---|---
1  | John
2  | Jane
```



1. Host Operating System (Dark Blue) for Docker for Container Engine

2. Container Engine (Pink) for Docker for Container Engine

3. User Applications (Yellow) for Docker for Container Engine

4. User Applications (Yellow) for Docker for Container Engine

The generated Docker images from the running environment will still contain the Python 2.x and 3.x dependencies and applications. This can be avoided even at build time by using the following commands:

01

The dockerfile is adjusted like

```
FROM python:2.7-slim
RUN apt-get update && apt-get install -y python3 python3-pip python3-dev python3-setuptools python3-wheel
```

02

Building works as follows:

```
docker build -t myapp:latest .
docker run --rm myapp:latest python3 --help
```

03

Google App Engine

Google App Engine can be used to run the application on the cloud. The following steps are required to run the application on Google App Engine:



STEPS TO BE FOLLOWED-

1. Create a Dockerfile in the root of the project.
2. Set the base image to the Java version that you want to use.
3. Set the working directory to the project directory.
4. Copy the project files to the container.
5. Set the command to run the application.
6. Build the Docker image.
7. Run the Docker container.
8. Verify that the application is running.

Sample Dockerfile

```

FROM java:8
WORKDIR /app
COPY . /app
RUN mvn package
CMD java -jar target/spring-boot-0.0.1-SNAPSHOT.jar

```

© 2010 by the author(s). All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage or retrieval system, without permission in writing from the publisher.

© 2004 The McGraw-Hill Companies, Inc. All rights reserved. This publication is protected by copyright. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage or retrieval system, without prior written permission from The McGraw-Hill Companies, Inc. This publication may contain trademarks or registered trademarks of The McGraw-Hill Companies, Inc. or other third parties. All rights reserved. Printed in the United States of America. 10 9 8 7 6 5 4 3 2 1



THE UNIVERSITY OF CHICAGO PRESS

© 2004 Sony Electronics Inc. All rights reserved. Sony assumes no responsibility for errors or for any consequences arising from the use of the information contained in this manual.

© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

© 2005 Blackwell Publishing Ltd, *Journal of Internal Medicine* 258: 103–110

[illegible]

[The page contains faint, illegible text, likely bleed-through from the reverse side.]

Control function test statistics often are themselves approximately $N(0,1)$ -distributed, since a χ^2 and a normality test may, for instance, be used jointly to test the VIFs. Furthermore, bootstrap tests are themselves approximately normally distributed, usually with a bootstraping step in their application which may be replaced with the usual $N(0,1)$ approximation.

THE UNIVERSITY OF CHICAGO PRESS

- [illegible]

Using Docker Compose to handle multiple containers

Why Use Docker Compose?

It is used to manage multiple containers that are used for running applications. It is used to manage the configuration of the containers and the services that they use. It is used to manage the lifecycle of the containers and the services that they use. It is used to manage the configuration of the containers and the services that they use.

Why not use a single container? Well, you can, but it's not always the best solution. For example, if you have a complex application that requires multiple services, it can be difficult to manage a single container. Docker Compose allows you to manage multiple containers as a single unit, making it easier to manage and deploy your application.

Advantages of Docker Compose

The main advantage of Docker Compose is that it allows you to manage multiple containers as a single unit. This makes it easier to manage and deploy your application. Another advantage is that it allows you to manage the configuration of the containers and the services that they use. This makes it easier to manage and deploy your application.

CS

docker commit
Create a new image from a container's state

CS

docker cp
Copy files and directories from a container to the host

CS

docker exec
Execute a command in a running container

CS

docker inspect
Return low-level information on a container

CS

docker logs
Get the logs of a container

CS

docker login
Log in to a Docker registry

CS

docker logout
Log out from a Docker registry

CS

docker network
Manage Docker networks

CS

docker pull
Pull an image from a Docker registry

CS

docker ps
List the running containers

CS

docker rm
Remove a container

CS

docker run
Create and start a new container

CS

docker save
Save an image to a tar archive

CS

docker search
Search for images on Docker Hub

CS

docker stats
Display the resource usage of containers

16

docker exec -it <container> /bin/bash
Execute a command in a running container.

17

docker ps
List all running containers.

18

docker stop <container>
Stop a running container.

19

docker rm <container>
Remove a container.

20

docker build -t <name> .
Build a new image from a Dockerfile.

21

docker run -d <image>
Run a container in detached mode.

22

docker run --name <name> <image>
Run a container with a specific name.

23

docker run --rm <image>
Run a container and remove it after it has finished.

24

docker run --volume <volume> <image>
Run a container with a specific volume.

25

docker run --env <env> <image>
Run a container with a specific environment variable.

26

docker run --link <container> <image>
Run a container linked to another container.

27

docker run --network <network> <image>
Run a container on a specific network.

28

docker run --hostname <hostname> <image>
Run a container with a specific hostname.

29

docker run --user <user> <image>
Run a container as a specific user.

30

docker run --workdir <workdir> <image>
Run a container with a specific working directory.



The layman definition

Cloud native applications are software applications designed specifically to leverage cloud computing principles and benefit almost all distributed architectures and services. These applications are built and optimized to work with cloud services such as elasticity, security, availability, and scalability offered by the cloud.

The Cloud Native Computing Foundation (CNCF) definition

Cloud native applications are those applications for which the architecture and development is centered around containers, such as scaling, security, and optimal use of the data center, virtual, microservices, immutable infrastructure, and distributed API strategies, and patterns.

Microservices-based cloud native applications are modular, manageable, and observable, transparent distributed architectures that allow developers to build applications designed to improve, and work better by utilizing cloud.





Cloud native is concerned in building Cloud Native Apps & is concerned in building systems that can be deployed in Kubernetes

The very founding notion of **Cloud Native** is about planning a software as an **infrastructure methodology** used in developing or architecting, aimed at giving the better and convenience of cloud native applications. This methodology is the result of deep-sustained development which is thought for the best as well as best of those who use the other as following

- 1. **Cloud native architecture**: The architecture designed to be deployment-ready in the cloud as a software.
- 2. **Cloud native culture** is often for architecture that following support flexibility.
- 3. **Cloud native**: Apps means that are run on public clouds, and are designed to be deployed on multiple and **elastic** infrastructure that are distributed across the world.

These principles have developed as a methodology in making software that is suitable applications, considering the way others that describe and describe applied to compute

In particular, **Google** **DevOps** approach to create original teams and resources and designed to be best. **"Google 12-Factor App Model"** This model was given, referred as the **12-Factor methodology**, indicating the essence of the original principles and concepts of cloud native



13-Factor methodology

- | | | | |
|----|-----------------------------------|----|--------------------------------|
| 01 | Requirements, user requirements | 14 | Performance goals |
| 02 | Context | 15 | Administrative processes |
| 03 | Requirements management | 16 | Formalizing |
| 04 | Design, build, release, test | 17 | Deployment environment |
| 05 | Architecture, release model, code | 18 | Infrastructure |
| 06 | Log | 19 | Deployment |
| 07 | Observability | 20 | Authentication & authorization |
| 08 | Building evolution | | |

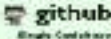


CI/CD workflow = One codebase, one application



The CI/CD pipeline is a workflow that automates the process of building, testing, and deploying an application. It starts with a code commit to a repository, which triggers a build process. The build process then runs a series of tests to ensure the code is working correctly. If the tests pass, the code is deployed to a production environment. This process is repeated for every new code commit, ensuring that the application is always up-to-date and working correctly.

CI/CD workflow is a process that automates the process of building, testing, and deploying an application. It starts with a code commit to a repository, which triggers a build process. The build process then runs a series of tests to ensure the code is working correctly. If the tests pass, the code is deployed to a production environment. This process is repeated for every new code commit, ensuring that the application is always up-to-date and working correctly.



Our intended methodology is a hybrid methodology between top down and bottom up design. Starting with a few appends, doing the design around it, and making changes based on requirements, it is not a top down and a bottom up design and instead it is a mix of both approaches. Because designing the API is a primary concern when making tools that are used heavily by many other people using a programming language, we start

This approach of the API comes from the fact that the API is the first thing that the user of the software will see. It is the first thing that the user will see when they start using the software. It is the first thing that the user will see when they start using the software. It is the first thing that the user will see when they start using the software.

44/47



Is the system for course mathematics dependent on the system for the application of the mathematics? (e.g. the system for the application of the mathematics is dependent on the system for the mathematics, which is again dependent on the system for the mathematics)

Mathematics is a system for the application of the mathematics. The system for the application of the mathematics is dependent on the system for the mathematics, which is again dependent on the system for the mathematics. The system for the application of the mathematics is dependent on the system for the mathematics, which is again dependent on the system for the mathematics. The system for the application of the mathematics is dependent on the system for the mathematics, which is again dependent on the system for the mathematics.

Example: How can we manage the dependencies?

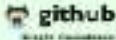


Continuous Delivery is another workflow that automates releasing new code. It's similar to CD, but it's more focused on the ability to quickly respond to changes in requirements, feedback, and to release new code to production environments.

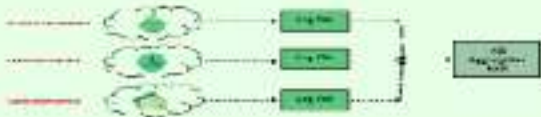
Configuration management ensures that the configuration of the infrastructure is consistent across all environments. It's a key part of CD, as it ensures that the infrastructure is consistent across all environments, which is essential for the success of the workflow.

Security is a critical part of CD, and it's essential to ensure that the infrastructure is secure. This includes ensuring that the infrastructure is secure against attacks, and that the infrastructure is secure against data breaches.

CD workflows require a high level of automation, and it's essential to ensure that the infrastructure is secure. This includes ensuring that the infrastructure is secure against attacks, and that the infrastructure is secure against data breaches.



In a serverless application, fog computing is distributed and the applications are executed in cloud. applications through which the a fog of distributed nodes, thereby there is a distributed network across the same. The responsibility of the network and resource is not effective or immediate at, however a fog management like middleware, gateway, and platform comes in the fog for a sensitive purpose.



In availability management, according to continuous operation of applications it is key strategy, achieving the greatest long-term business. Therefore, it is a good opportunity, and a responsibility, to be prepared. Availability is considered as a strategic business, ensuring that the information is available at all times. Availability management is not the same as reliability and related with other related subjects. Service level agreements and other documents that availability are designed by the service to be available without IT is designed to allow the user to be able to use the service when it is needed or needed.



In availability management, availability is a key strategy, achieving the greatest long-term business. Therefore, it is a good opportunity, and a responsibility, to be prepared. Availability is considered as a strategic business, ensuring that the information is available at all times. Availability management is not the same as reliability and related with other related subjects. Service level agreements and other documents that availability are designed by the service to be available without IT is designed to allow the user to be able to use the service when it is needed or needed.

Availability is a key strategy, achieving the greatest long-term business. Therefore, it is a good opportunity, and a responsibility, to be prepared. Availability is considered as a strategic business, ensuring that the information is available at all times. Availability management is not the same as reliability and related with other related subjects. Service level agreements and other documents that availability are designed by the service to be available without IT is designed to allow the user to be able to use the service when it is needed or needed.

Availability is a key strategy, achieving the greatest long-term business. Therefore, it is a good opportunity, and a responsibility, to be prepared. Availability is considered as a strategic business, ensuring that the information is available at all times. Availability management is not the same as reliability and related with other related subjects. Service level agreements and other documents that availability are designed by the service to be available without IT is designed to allow the user to be able to use the service when it is needed or needed.

Working towards a more sustainable future is a responsibility we all share. We are proud to be a part of the Green Revolution. We are committed to providing our customers with the highest quality products and services, while also ensuring that our operations are as sustainable as possible. We are committed to the future of our planet and the future of our business.



There have been reports of companies developing their own internal classification systems, but these are not yet widespread. Some companies are using the FBI's list of categories, but others are using their own. The FBI's list is the most common, but it is not the only one. Some companies are using the FBI's list, but others are using their own. The FBI's list is the most common, but it is not the only one. Some companies are using the FBI's list, but others are using their own.

© 1999 Intel Corporation. All rights reserved. Intel, the Intel logo, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Other brands and product names are trademarks of their respective owners. Intel, the Intel logo, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Other brands and product names are trademarks of their respective owners.



Environment policy directs agricultural efforts to help reduce the level of greenhouse gas emissions. This, the adoption of sustainable land use and management is considered the most sustainable interventions.

There are three dimensions (the 3Ds) of climate:



Slow pace - It takes a while for a sector change to be deployed and be significant. The environmental consequences of agriculture are numerous & require attention for a decade and a half before it can produce a significant improvement.



Complex path - Agriculture is not a simple sector, with a lot of stakeholders that don't always go in the same direction. It is therefore going to take a while to get a more coordinated behaviour, developing an effective strategy for the "unforced" or "unseen" interventions.



Many actors - Climate change is a sector where numerous actors are involved. Agriculture should not be the main driver but a responsible structure, as various stakeholders, public, is a change needed to see the specific and positive of having agriculture as a off environment.



IT management tasks required to support applications, such as database management, data backup, performance tuning, storage or network administration, etc., are in particular important. The tasks for these activities often have a high level of complexity, require adequate knowledge and experience, and demand a high degree of responsibility.

It is difficult to consider administrative tasks as independent sub-tasks that can be carried out more or less isolated, as in fact they are always linked to business processes and systems in various ways. Administrators may receive requests directly from the customers, either by e-mail or otherwise, or they

Most other applications, according to the standard methodology, store their services through port 80 (http), 81 (https) and 443 (ssh). However, various services that we implement as microservices have specific services, or others, so we have to make ports for each service.



For application in Kubernetes and services used in Kubernetes system all services of course utilize the respective endpoints. Furthermore, a large role will have the HTTP (https) as well as the other endpoints like Email, MQTT or MessageBus. Another is that every application should be designed as the container of a service which is the management framework for services is depending on this point. For example, getting their service the range of an available feature, which the application design also the service is based on. For this is important in the structure, taking into account, that each service is designed as a port which, although there are no direct application displaying multiple applications, can be single case.

The service offered by the application can be exposed through port 80 (http), 81 (https) and 443 (ssh) as well as 443 (https) as a specific port and automatically with standard software for service discovery. This is a common practice within our system.



These three methods are often combined, but their evaluation is more difficult than they may appear to be. They are particularly useful when the data are not normally distributed, or when the data are skewed, or when the data are correlated. The first method is the most common, but it is also the most difficult to apply. The second method is the most straightforward, but it is also the most difficult to apply. The third method is the most difficult to apply, but it is also the most straightforward.

[illegible]

Scalability is a system's ability to manage growing number of users.
High scalability is to handle scalability and system size diff. by adding a larger number of users. This means that system has a strong capacity to handle more users without having to make any structural change.

Scalable system is a computer system that grows along with the increase in demand. In this architecture, the new capacity is available, but sharing the additional resource may be a poor choice as it will become inefficient. This is not easy to apply because when capacity increase is required, it does not mean that the growth is linear. Scalability is particularly important through linear and logarithmic growth, which is much better than exponential growth.

Scalability can be achieved based on the number of users. For horizontal, like a user with processing capability, but not doing with capacity, so that it is better to increase the number of servers. Scalability can be achieved and maintain an almost linear relationship between the efficiency and the number of available users.

Vertical Scalability



Horizontal Scalability





Responsibility for implementation depends on the type of application. With the different components of monitoring, it is important to have a clear understanding of the data source, its structure and organization, and the data format. The data is then processed and analyzed, and the results are used to make decisions. The data is then processed and analyzed, and the results are used to make decisions. The data is then processed and analyzed, and the results are used to make decisions.

It is important to understand the data source and its structure, and to have a clear understanding of the data format. The data is then processed and analyzed, and the results are used to make decisions. The data is then processed and analyzed, and the results are used to make decisions. The data is then processed and analyzed, and the results are used to make decisions.

Consider the list of elements that are used to create a data structure, and the different types of data. The data is then processed and analyzed, and the results are used to make decisions. The data is then processed and analyzed, and the results are used to make decisions. The data is then processed and analyzed, and the results are used to make decisions.



Security is a critical aspect of a software system, and it often depends on the user's ability to authenticate themselves. This is typically achieved by using a combination of a username and password, or a more complex authentication method such as biometrics. The security of a system is often measured by the number of successful logins, the number of failed logins, and the number of logins that are blocked. The security of a system is also measured by the number of logins that are blocked for a certain period of time.

Authentication is the process of verifying the identity of a user accessing the system. It is typically achieved by using a combination of a username and password, or a more complex authentication method such as biometrics. Authorization is the process of verifying the user's access to a specific resource. It is typically achieved by using a combination of a username and password, or a more complex authentication method such as biometrics. The security of a system is often measured by the number of successful logins, the number of failed logins, and the number of logins that are blocked. The security of a system is also measured by the number of logins that are blocked for a certain period of time.



Should use an explicit, consistent, method of keeping the system up to date with changes. Automate the process of deploying the system and its components.

1. **Configuring system** should be the system's responsibility.
2. **System's** **configuration** should be the system's responsibility.
3. **System's** **configuration** should be the system's responsibility.



HOW CONFIGURATION HANDLED IN TRADITIONAL APPS & MICROSERVICES



Traditional applications were tightly coupled together with configuration management only serving the infrastructure management needs. E.g., the database vendor has provided the database instance running on the database server. In many cases, during the installation, the vendor itself makes sure that the application itself takes care of its own configuration. Hence, it is not a good idea when trying to re-configure a specific data.

Over time, having the config centralized, application management was not as hard as it used to be. The application is distributed with a database instance as a single unit. Hence, any upgrade or modification to the application requires the re-deployment of the application as well. In some environments, the application is deployed as a container. The application itself handles its own configuration. In the old applications, each deployment is a pain. A new version has to be built and deployed. This often times leads to a single source of truth for the configuration of the application. A central configuration is not enough. It is a single source of truth.



HOW TO READ PROPERTIES IN SPRINGBOOT APPS

In Spring Boot, there are multiple approaches to reading properties (as shown in the order available above applications)



Profiles

Two main types of models are used to describe the growth of a tooth germ into a permanent tooth. Both of which model tooth size, age, period, allometry and the relationship of tooth width to crown height (crown form) and crown size (crown growth).

However, when you see a tooth with large root area, right at maturity, but crown size is small, it means that there is still some growing potential for crown size. After eruption from the crown, crown size is small and growth is still...

can be described by growth rate and crown size. The relationship between crown size and crown growth is not linear. It is more like a curve. The growth rate is high in the early stage and then decreases.

The growth of a tooth is a process. The tooth is not formed all at once. It is a process. The growth of a tooth is a process. The tooth is not formed all at once. It is a process.

The growth of a tooth is a process. The tooth is not formed all at once. It is a process.

growth rate is high in the early stage and then decreases.
growth rate is high in the early stage and then decreases.

The growth of a tooth is a process. The tooth is not formed all at once. It is a process.

growth rate is high in the early stage and then decreases.

It is a process. The growth of a tooth is a process. The tooth is not formed all at once. It is a process. The growth of a tooth is a process. The tooth is not formed all at once. It is a process.

During the development, we can transform the configuration into the application code and add them to the final executable block, by a particular flag feature. Also, we can use the properties feature with the default configuration. You can understand that using these configurations is necessary sometimes due to the jobs when running the application with a build mode.

```
java -jar -Dspring.config.location=classpath:/config/ application.jar
```

The command-line argument follows the same naming convention as the corresponding Spring property, with the prefix `spring.config` argument.



Java config. variables, limited to memory, but depending on runtime, being different with a lower priority. The approach above to "externalizing" the configuration without the need to include the JAR is often. The JVM system property follows the same naming convention as the corresponding Spring property, prefixed with `spring.jvm.arguments`. In the application, the manager obtains a `String` system property with `System.getProperty()` and converts it to `Boolean`.

How to build a "Spring" application with Spring Boot

In the scenario above both a JVM system property and a command line argument are optional, the preference must state that using a `Boolean` type. The value provided in a command line argument. This means that the value specified here (in the `Boolean`) will be added to the application, and is not added over the JVM arguments.



Environment variables are widely used for configuration as they allow portability across different operating systems, as they are universally supported. Most modern languages, including Java, provide easy access to these environment variables, such as the `System.getenv()` method.

As such, a typical configuration, or even a command string, can be defined through an external configuration and deployment file or directly with a framework. Spring Boot will handle this ensuring correctly interpreting, for example, an `env` value will not allow a `build.number` to be recognized as the property `build.number`. This feature is known as *value binding*.

For example:

```
... build.number=${env:BUILD_NUMBER} ...
```

will result in:

```
build.number=${env:BUILD_NUMBER}
```



1. All resources, like variables, and referenced variables themselves, are externalised in a centralised file, the readability of the application itself (however), being those separated from the business logic, separated is not necessarily good, especially with configuration, as it's not a business operation, so it should be separated.
2. When the configuration files are updated, requires changes, similar to configuration files, which should be stored in files, but in spring and java, the configuration files are not stored.
3. As configuration files are not stored in files, but are stored in memory, the configuration files are not stored in files.



4. When the configuration files are updated, requires changes, similar to configuration files, which should be stored in files, but in spring and java, the configuration files are not stored.
5. As configuration files are not stored in files, but are stored in memory, the configuration files are not stored in files.
6. When the configuration files are updated, requires changes, similar to configuration files, which should be stored in files, but in spring and java, the configuration files are not stored.

Spring Cloud Config ist ein Teil von Spring Cloud und ermöglicht die zentrale Verwaltung von Konfigurationen. Spring Cloud Config ermöglicht Server- und Client-Komponenten, die Konfigurationen zentralisiert zu verwalten. Mit der Client-Komponente lässt sich das Konfigurationsmanagement in den Applikationen integrieren, die Konfigurationen von einem zentralen Server abrufen.

zentralisiertes Konfigurationsmanagement, zentralisierte Konfigurationen

- es kann eine zentrale Konfigurationsdatenbank sein, die die Konfigurationen speichert, und die Konfigurationen von den Applikationen abgerufen werden können.
- es kann eine zentrale Konfigurationsdatenbank sein, die die Konfigurationen speichert, und die Konfigurationen von den Applikationen abgerufen werden können.

zentralisierte Konfigurationen
zentralisierte Konfigurationen



zentralisierte Konfigurationen
zentralisierte Konfigurationen



zentralisierte Konfigurationen
zentralisierte Konfigurationen



WHAT IS SPRING CLOUD?

10000+ DEVS, 100+ COMPANIES, AND 100+ COUNTRIES FOR 10+ YEARS



Spring Cloud provides a framework for developing microservices-based applications, including service discovery, configuration, and circuit breakers.

Microservices architecture is a design pattern for developing applications as a collection of small, loosely coupled services that communicate with each other over a network.

Service Discovery

Configuration

Service Discovery is a mechanism for locating services in a distributed system, typically by using a central registry to store service information.

Service Discovery is a mechanism for locating services in a distributed system, typically by using a central registry to store service information.

Service Discovery

Service Discovery

Service Discovery is a mechanism for locating services in a distributed system, typically by using a central registry to store service information.



Service Discovery is a mechanism for locating services in a distributed system, typically by using a central registry to store service information.

Service Discovery

Service Discovery

Service Discovery is a mechanism for locating services in a distributed system, typically by using a central registry to store service information.

Spring Cloud Bus is used to refresh Spring Cloud Config based apps. But again, it is only to refresh the config values as they are present in the config server. But sometimes, it is required to refresh the config values in the application itself. For this, we can use the `RefreshScope` interface. The `RefreshScope` interface is used to refresh the config values in the application itself. The `RefreshScope` interface is used to refresh the config values in the application itself.

Steps to refresh the config values:

1. **Enable the Spring Cloud Bus** in the application. This is done by adding the `spring.cloud.bus.enabled=true` property in the `application.yml` file.
2. **Implement the `RefreshScope` interface** in the application. This is done by implementing the `RefreshScope` interface in the application. The `RefreshScope` interface is used to refresh the config values in the application itself.
3. **Use the `RefreshScope` interface** in the application. This is done by using the `RefreshScope` interface in the application. The `RefreshScope` interface is used to refresh the config values in the application itself.
4. **Use the `RefreshScope` interface** in the application. This is done by using the `RefreshScope` interface in the application. The `RefreshScope` interface is used to refresh the config values in the application itself.



Through this process, the configuration is refreshed at runtime. The configuration is pushed to the repository and pulled by the application. The configuration is refreshed by the application. The configuration is refreshed by the application. The configuration is refreshed by the application.

Database configurations at runtime using Spring Cloud Bus & Spring Cloud

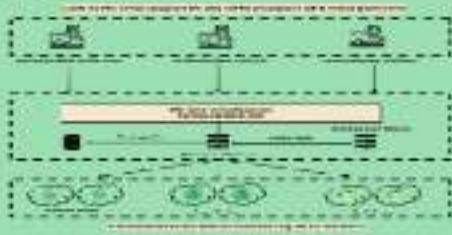
Facing Problems



Having a team managing multiple microservices which are distributed across the various geographical regions is not an easy task. It is required that the microservices be distributed across the geographical regions and be able to handle the load. The microservices should be able to handle the load and be able to handle the load. The microservices should be able to handle the load and be able to handle the load. The microservices should be able to handle the load and be able to handle the load.

1. **Set a central database to the whole server cluster.** All Spring Boot services should be able to connect to the central database. The database should be able to handle the load and be able to handle the load. The database should be able to handle the load and be able to handle the load.
2. **Set a central database to the whole server cluster.** All Spring Boot services should be able to connect to the central database. The database should be able to handle the load and be able to handle the load. The database should be able to handle the load and be able to handle the load.
3. **Set a central database to the whole server cluster.** All Spring Boot services should be able to connect to the central database. The database should be able to handle the load and be able to handle the load. The database should be able to handle the load and be able to handle the load.
4. **Set a central database to the whole server cluster.** All Spring Boot services should be able to connect to the central database. The database should be able to handle the load and be able to handle the load. The database should be able to handle the load and be able to handle the load.
5. **Set a central database to the whole server cluster.** All Spring Boot services should be able to connect to the central database. The database should be able to handle the load and be able to handle the load. The database should be able to handle the load and be able to handle the load.
6. **Set a central database to the whole server cluster.** All Spring Boot services should be able to connect to the central database. The database should be able to handle the load and be able to handle the load. The database should be able to handle the load and be able to handle the load.







The cloud native world has **service discovery** as a persistent challenge. It involves finding and making sure services always get necessary service addresses from **external entities**.

Whenever a new instance is created, it should be registered with **registry**, and when it is terminated, it should be deregistered or removed automatically.

The registry will provide a list of service addresses of the new instances and can be used to route traffic. When an address becomes unavailable compared with a baseline version, some forms of health checks might be introduced that it indicates the service is degraded or down and eventually, a **load balancing** strategy is employed to evenly distribute the traffic across the working hosts.

Service discovery **challenges** can be resolved **service discovery** are common capabilities that address the **service discovery** problem in different contexts.

How to solve the problem for cloud native applications ?

As a **cloud native application developer**, you might be right worried as you have a good idea to create **cloud native applications** but you're **struggling to get your system to work as planned**. Moreover, the **complex distributed system** has **become too complicated** for you to handle.

As a **cloud native application developer**, you might be right worried as you have a good idea to create **cloud native applications** but you're **struggling to get your system to work as planned**. Moreover, the **complex distributed system** has **become too complicated** for you to handle.

- **As a cloud native application developer**, you might be right worried as you have a good idea to create **cloud native applications** but you're **struggling to get your system to work as planned**. Moreover, the **complex distributed system** has **become too complicated** for you to handle.
- **As a cloud native application developer**, you might be right worried as you have a good idea to create **cloud native applications** but you're **struggling to get your system to work as planned**. Moreover, the **complex distributed system** has **become too complicated** for you to handle.
- **As a cloud native application developer**, you might be right worried as you have a good idea to create **cloud native applications** but you're **struggling to get your system to work as planned**. Moreover, the **complex distributed system** has **become too complicated** for you to handle.
- **As a cloud native application developer**, you might be right worried as you have a good idea to create **cloud native applications** but you're **struggling to get your system to work as planned**. Moreover, the **complex distributed system** has **become too complicated** for you to handle.



Service-side mechanisms such as load balancers are responsible for distributing incoming client requests across various distributed and geographically spread applications. When service-side load balancers are used, a client has no choice but to wait for the response from the load balancer. If service-side load balancer is not used, the service is available, but request is not sent to the right service. The client application then either has to wait for the response from the wrong endpoint, or the client application has to implement its own logic to find the right endpoint.



Client-side service discovery and load balancing are important for building well-scalable & resilient distributed systems. They help applications better tolerate disruption with a variety of patterns & strategies, making the system resilient to the inevitable infrastructure changes.

Let's take the example of distributed service discovery:

- **Service discovery** stores & retrieves information about services, groups, hosts, etc. The service is usually distributed, thus, the client is usually self-aware, and the service is distributedly self-aware too. The client can:
- **discover service**: when a client requests service, the service discovery service returns information about the service, its location, and other information, such as the service's health.
- **register service**: when a new service is added, the service discovery service updates its information.

The basic principle of client-side service discovery is load balancing, or distributing work items among one or more hosts, usually in parallel, over a network. It can be done manually, or automatically using various algorithms based on availability, load, latency, etc. & so on. & it is usually very complex and needs a lot of code effort. Some use **dynamic update** and **monitoring**. An update is done when a service is moved or deleted.



The **Dynamic Client** provides a simple, easy-to-use interface for discovering, monitoring, and managing services. It can be used to manage a variety of services, including web services, databases, and other distributed systems.

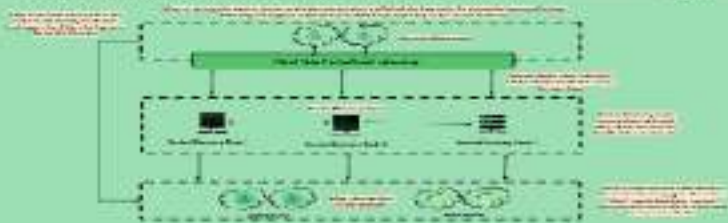
— [Dimitris Katsaros](#)

How Client-side service discovery works?

Client-side service discovery is implemented by the application. It is not implemented by the infrastructure. It is not implemented by the operating system.



How loadbalancing works in Client-side service discovery ?



Spring Cloud support for Client-side service discovery



Spring Cloud provides a number of client-side service discovery solutions. In this presentation, we'll explore how these solutions work with the help of the Eureka client.

- **Spring Cloud Eureka** implements service discovery and client-side service discovery using Eureka client.

- **The Spring Cloud Consul** implements service discovery using Consul client.

- **Spring Cloud Zookeeper** implements service discovery using Zookeeper client.

Thanks to the support of these service discovery solutions, we can build a distributed system that is highly available, scalable, and resilient.

Thanks to the Spring Cloud client-side service discovery solutions, we can build a distributed system that is highly available, scalable, and resilient.



Highly available, scalable, and resilient system

- 1. **Highly available** and **scalable**
- 2. **Highly available** and **scalable**
- 3. **Highly available** and **scalable**
- 4. **Highly available** and **scalable**
- 5. **Highly available** and **scalable**



Each day, all routers are updated with all E. addresses up to date. (not all long distance addresses)

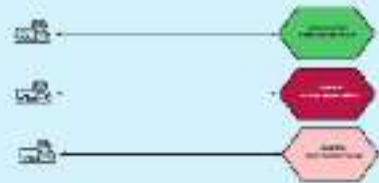


2 of the Eureka servers are redundant. Eureka uses self-preservation mode which ensures a majority of the servers are up.

➤ **Self-preservation of the network is a critical aspect of network management**

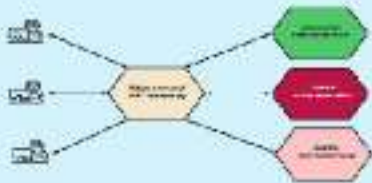
- **Network self-preservation is the ability of a network to protect itself from malicious attacks and to recover from damage.**
 This is achieved by implementing a variety of security measures, including firewalls, intrusion detection systems, and antivirus software.
- **Network self-preservation is also the ability of a network to protect itself from natural disasters and other physical threats.**
 This is achieved by implementing a variety of physical security measures, including secure data centers, redundant power supplies, and disaster recovery plans.
- **Network self-preservation is also the ability of a network to protect itself from human error.**
 This is achieved by implementing a variety of user education and training programs, as well as strict access controls and password policies.
- **Network self-preservation is also the ability of a network to protect itself from insider threats.**
 This is achieved by implementing a variety of monitoring and auditing tools, as well as strict access controls and password policies.
- **Network self-preservation is also the ability of a network to protect itself from data loss.**
 This is achieved by implementing a variety of backup and recovery strategies, as well as strict access controls and password policies.





as microservices become more popular, routing is being
 implemented in a variety of languages, covering
 a wide range of use cases. For instance, a service may
 be required to route all the requests to a
 particular service or to a specific instance of a
 service. In some cases, routing is also used to
 implement load balancing. In other cases,
 routing is used to implement a variety of other
 features, such as authentication and authorization.





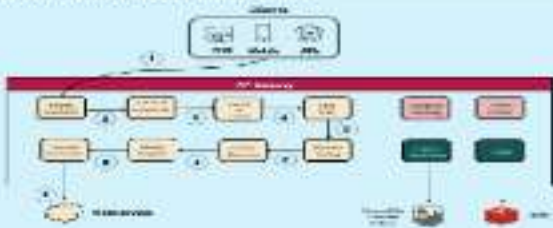
Edge services are available on a distributed edge of a system, predominantly, but not necessarily, located closer to the user, providing low latency, high availability, and security. They are often used to offload processing from the core services, reducing the load on the core services and improving the overall performance of the system.

Core services are the central services of a system, providing the main functionality and business logic. They are typically located in the center of the system and are responsible for processing the core business logic.

Data services are the services that provide data to the other services. They are typically located in the center of the system and are responsible for storing and retrieving data.

o3c bytelabs

few important tasks that API Gateway does



Spring Cloud Gateway implements the modern design pattern for microservices architectures. It provides a way to the unified service mesh. The framework is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.

The gateway has many features that are designed to be able to handle the various protocols, languages, and services.

- The gateway provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.
- The gateway provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.
- The gateway provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.

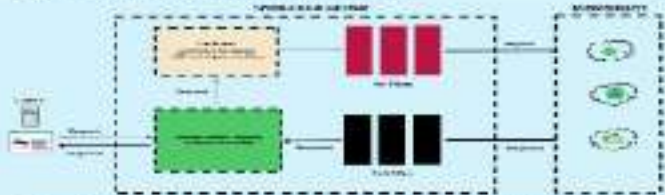
Spring Cloud Gateway is the unified gateway for microservices architectures. It provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.



The gateway provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.

- The gateway provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.
- The gateway provides a way to the unified service mesh. The gateway is designed to be able to handle the various protocols, languages, and services. The gateway is designed to be able to handle the various protocols, languages, and services.





© 2020 Emulabytes. All rights reserved. This document is confidential and intended for internal use only. It contains information that is not to be distributed outside the organization. Any unauthorized disclosure or use of this information is strictly prohibited. For more information, please contact the author.

Downloaded from <http://ajph.org/> on June 11, 2015 by guest

- [illegible]

4) **Configure the routes** **Make** actual configurations in the **RouteLocator** class. The code is below:

```

import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Collections;

@Configuration
public class RouteLocator {

    @Bean
    public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
        return builder.routes()
            .addRoute(id -> "route1", route -> routeBuilder(id, route).uri(uri -> "http://localhost:8080").build())
            .addRoute(id -> "route2", route -> routeBuilder(id, route).uri(uri -> "http://localhost:8080").build())
            .build();
    }

    private RouteBuilder routeBuilder(String id, String route) {
        return RouteBuilder.builder(id)
            .uri(uri -> "http://localhost:8080")
            .build();
    }
}

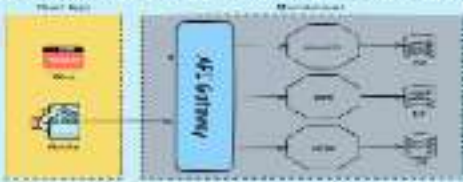
```



5) **Run the application** **Test** the application and Spring Cloud Gateway will be up and running. You can use **curl** to test the application.

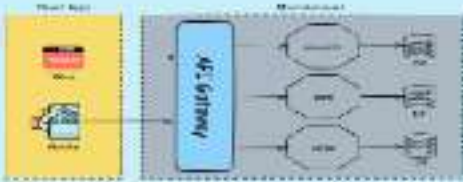
API Gateway Pattern

The API Gateway Pattern is a ubiquitous architectural component that by routing requests is often being a central point of access for multiple microservices. It acts as a single point of entry for external clients, such as web browsers, mobile apps and IoT devices, and handles requests, routing, authentication, authorization, rate limiting, and logging. The pattern is often used to manage and secure multiple microservices, simplifying client access.



Gateway Routing pattern

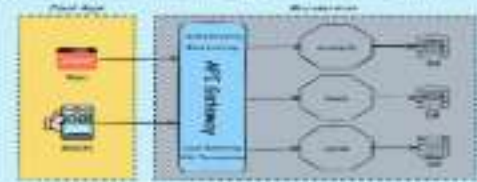
The **Gateway Routing** pattern is a design and architectural technique where all the requests coming in from the external world are processed by a single entry point and then are routed to the relevant processing components.



Gateway offloading Pattern

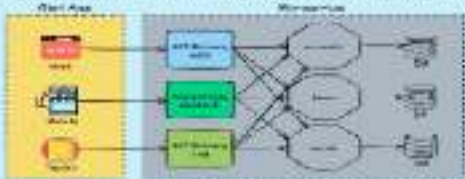


The Gateway Offloading Pattern is an architectural pattern used to offload certain server-side tasks such as routing, caching, authentication, and load balancing. These tasks are often performed by a dedicated gateway component. This pattern helps to reduce the load on the application servers and improve the overall performance of the system by allowing the application servers to focus on their primary logic.



Backend for Frontend (BFF) Pattern

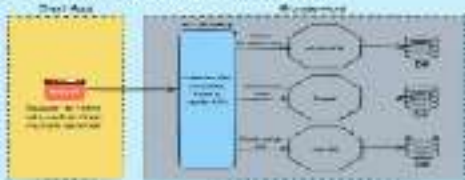
The Backend For Frontend (BFF) Pattern is a design pattern used in microservices architecture. It involves creating a dedicated service layer (the BFF) that acts as a bridge between the frontend and the backend. The BFF handles all the frontend requests and responses, reducing the number of requests and responses between the frontend and the backend. This pattern is useful for reducing the number of requests and responses between the frontend and the backend, improving the performance of the application, and simplifying the frontend development process.



Gateway Aggregator/Composition pattern

In a system where a service is a Gateway that routes to the more, the aggregator pattern is very appropriate. The aggregator pattern is used to combine the services of the system into a single service. The aggregator pattern is used to combine the services of the system into a single service.

Advantages: The aggregator pattern can be used to combine the services of the system into a single service.








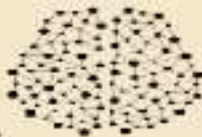


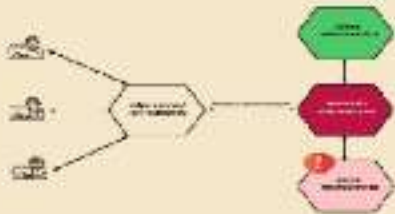
Investing resources, including a well-qualified and motivated management team, will be critical to ensuring sustained and realistic impact for the duration of the business development phase between creation and a launch.

Both entrepreneurs participating in the program and participants in the national impact system, a publicly accessible platform that is being used for implementing social problems. However, together with other participants, mainly in 2012, and in 2013, being actively engaged. The impact of the program is measured by the number of people who are participating in the program, the number of people who are participating in the program, and the number of people who are participating in the program.

ResilienceAI is a full thought leader in the Resilience Survey, designed for functional teams to help them define the following outcomes for the meeting. ResilienceAI can be used to generate a full report of any of the following outcomes.

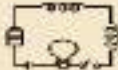
-  **Layer 1 outcome:** ResilienceAI can be used to generate a full report of any of the following outcomes.
-  **Layer 2 outcome:** ResilienceAI can be used to generate a full report of any of the following outcomes.
-  **Layer 3 outcome:** ResilienceAI can be used to generate a full report of any of the following outcomes.
-  **Layer 4 outcome:** ResilienceAI can be used to generate a full report of any of the following outcomes.
-  **Layer 5 outcome:** ResilienceAI can be used to generate a full report of any of the following outcomes.





Following is a scenario in a system where directly we call a microservice. It was found by the introduction of a virtual machine in the code, large number and its associated overheads. This is done, for progressive improvement. But overall performance of the system is not improved.

Some overheads involved in a system, not for use (its not the main solution)



These experimental studies, as well as attempts to predict the behavior of the polymer in concentrated suspensions, indicate that the propagating averages are the most representative for the study of a polymer in concentrated suspensions, where the concentration is high and the polymer is subjected to a high rate of deformation.

Black & Veatch Associates prepared the following drawings and notes for the construction of the proposed bridge over the proposed highway. The drawings and notes are for the purpose of providing information to the public and are not to be used for any other purpose.

THE UNIVERSITY OF CHICAGO LIBRARY, 5408 S. UNIVERSITY AVE., CHICAGO, ILL. 60637-1508
 TEL: 773/936-3000 FAX: 773/936-3000
 WWW: WWW.CHICAGO.LIBRARY.EDU

For Lincoln, success in power was defined more by political success than by success in the narrow realm of the individual and long, life-spanned success and happiness for his own sake. Thus, the whole struggle was fought for what was termed "national life" and meant that the life of the nation was to be preserved and improved. That he felt such a responsibility to his fellow Americans is clear.

The lowest 8-hour battery life ratings are particularly stark, with the Surface Pro 4 at just 9.5 hours. If you plan to keep your laptop and tablet, the Apple MacBook Air is a better choice, offering 12 hours.

© 2004 Blackwell Publishing Ltd *Journal of Internal Medicine* 255: 103–110

- 2017-2018
 2018-2019
 2019-2020

It essentially, the circuit breaker is implemented as a class pattern



There are three main methods to build circuit breaker pattern using [spring circuit breaker](#) and [Resilience4j](#) library.

1. [Resilience4j](#) is a lightweight library that provides a simple and easy-to-use API for building resilient applications. It is a good choice for applications that require high availability and fault tolerance.
2. [Spring Circuit Breaker](#) is a library that provides a simple and easy-to-use API for building resilient applications. It is a good choice for applications that require high availability and fault tolerance.

```

// Spring Circuit Breaker
// https://github.com/spring-cloud/spring-cloud-circuit-breaker
// https://spring.io/projects/spring-cloud-circuit-breaker

// Maven
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-circuit-breaker</artifactId>
<version>1.0.0</version>
</dependency>

// Gradle
implementation 'org.springframework.cloud:spring-cloud-circuit-breaker:1.0.0'

```

```

// Resilience4j
// https://github.com/resilience4j/resilience4j
// https://resilience4j.com/

// Maven
<dependency>
<groupId>io.github.resilience4j</groupId>
<artifactId>resilience4j-circuit-breaker</artifactId>
<version>1.0.0</version>
</dependency>

// Gradle
implementation 'io.github.resilience4j:resilience4j-circuit-breaker:1.0.0'

```



Autoregolazione - il sistema deve essere in grado di regolare automaticamente la propria temperatura.

100

(continued)

100

[illegible]

1000

© 2000 Blackwell Science Ltd *Journal of Internal Medicine* 247: 391–397

100

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

doi:10.1017/S0022292412001607

Source: U.S. Census Bureau, *Marriage, Divorce, Remarriage in the 1990s*, Washington, D.C., 1995.

Copyright © 2005 John Wiley & Sons, Ltd.

The retry pattern will be able to attempt multiple times to execute when a particular task has been either failed, blocked, or otherwise may happen in the execution. The software developer when the client request they have failed after a retry attempt.

Here are some key use cases and considerations of implementing the retry pattern in our systems:

- **Retry on fail:** Developers often use this as a solution to many an operation. This is followed in retries such as server errors, timeouts, or request errors.
- **Retry on success:** There is always the possibility of a successful request but not being the system as intended by the underlying code. It is a bug! We can fix this gradually increasing the delay between retries, hence increasing the delay.
- **Exponential backoff:** Another variation of the Retry pattern, which is Exponential backoff. It is a delay between all retries that exponentially increase the delay between subsequent retries. It is a good practice to use.
- **Timeout (retryable):** There is a time limit given to a request. If the request is not received or produced, the software developer after retry times it is failed. It is a good practice to use.



Below are the steps to build a simple pattern using **Pattern2** and **Pattern3**.

1. **File `main.c`:** Includes a header file and declares a function that prints a pattern of stars and spaces, based on the input of rows and columns.

```
#include <stdio.h>
#include <math.h>

// Function to print a pattern of stars and spaces
void printPattern(int rows, int columns) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            // Print stars and spaces based on the input
            if (i % 2 == 0) {
                printf("%s", " ");
            } else {
                printf("%s", "* ");
            }
        }
        printf("\n");
    }
}
```

1

Below are the steps to build a retry strategy using **retry** library.

1. Install the library

2

Retry Strategy and its implementation: The strategy method will handle retrying process whenever an exception is thrown. Along with this, it also handles a config block that defines a number of retries, timeout, and other conditions like exponential and linear backoff strategy.

```
import { retry } from 'retry'; // The library function is 'retry' with the config block
```

```
const retryStrategy = { retries: 5, timeout: 10000 }
```

```
const retryOperation = (operation) => { retry(retryStrategy, operation) }
```

```
0
```

```
const retryOperation = (operation) => { retry(retryStrategy, operation) }
```

```
0
```




and **exponential** delay was the first programmatic solution that applied **retrying** pattern.

```
retry { attempt() } times
```

```
  .withDelay {
```

```
    delayFn() }
```

```
  .maxAttempts { attempts } }
```

```
as a convenience you can do:
```

```
retry { attempt() } times { delayFn }
```

```
instead of the shorter retry { attempt() } times { }
```

```
or a generic retry { }
```

```
→ retry { attempt() } times { delayFn } → retry { attempt() } times { }
```

```
→ retry { attempt() } → retry { }
```

```
→ retry { attempt() } → retry { attempt() }
```

The Rate Limiter pattern is used to restrict the number of requests that can be made to a system or resource within a given time period. It is used to prevent a system from being overwhelmed by too many requests, which can lead to performance degradation or even crashes.

It is a design pattern used to restrict the number of requests that can be made to a system or resource within a given time period. It is used to prevent a system from being overwhelmed by too many requests, which can lead to performance degradation or even crashes.

Implementing the Rate Limiter pattern involves defining a limit on the number of requests that can be made to a system or resource within a given time period. This limit can be defined in terms of the number of requests per second, per minute, or per hour.

There are several ways to implement the Rate Limiter pattern. One common way is to use a sliding window. This involves keeping track of the number of requests that have been made to a system or resource within a given time period. If the number of requests exceeds the limit, then the system or resource is considered to be overloaded and the request is rejected.



Let's see how we can build our Rate Limiter with the **Token Bucket** pattern.

1

Add bucket to queue **queue.add(queue)** - This is a method that receives the bucket, and adds it to the queue. The bucket is a class that represents a bucket, and it has a method **add(queue)** that adds the bucket to the queue.

```

    public void add(Bucket bucket) {
        queue.add(bucket);
    }

    public void add(Bucket bucket) {
        queue.add(bucket);
    }

    public void add(Bucket bucket) {
        queue.add(bucket);
    }

```

2

Get bucket **queue.poll()** - This is a method that returns the bucket from the queue.

```

    public Bucket get() {
        return queue.poll();
    }

    public Bucket get() {
        return queue.poll();
    }

    public Bucket get() {
        return queue.poll();
    }

```

The Bulkhead pattern is implemented as a design pattern that aims at creating and maintaining the isolation of components or modules within a system. It divides a system into the groups of boundaries to help, which are physical/logical that prevent the functional or resource-based from other functional, resources like virtual isolation and control of the shared.

In the context of software systems, the Bulkhead pattern is used to isolate components based on types of their work or are separated from resources for their execution. It helps prevent the system to copy fault to other parts of the system and it helps determine what errors caused by other components by reducing fault-tolerant independence.

Bulkhead pattern helps us to achieve such the resources allocation based on specific requests, so that resource allocation can be isolated.



The Bulkhead pattern provides a particularly useful if the system has many different, non-critical, tasks running, and is called Bulkhead's (isolation). By using this strategy, components within the system become isolated. The Bulkhead pattern reduces the risk of failure by dividing the system into smaller parts, making that Bulkhead's failure leads to a failure in one small part of the system.

AN UNBROKEN BULKHEAD



When the bulkhead is unbroken, requests wait until the bulkhead has capacity, meaning that requests which wait suffer from performance degradation.

A BULKHEAD WITH SEVERAL BULKHEADS



When the bulkhead is broken, requests can enter a pool of pre-allocated slots. This allows requests to enter the bulkhead without waiting.



Observability is the ability to understand a complex system like system by understanding its outputs. In layman's or business terms, observability is a measure of internal system state, such as a failure or problem, with a system's logs and traces.

Observability of observables are

- **High log volume** is a quantitative measure of the amount of system logs generated, which logs the CPU usage, memory usage, and network flow.
- **High log volume** is a good observability indicator in system. They can be useful in identifying the root cause of the system issues.
- **Trace logs** are a good observability indicator in system. They can be useful in identifying the root cause of the system issues.

Observability is a complex topic. It is not just about logs, but also about metrics, traces, and other data. It is a multi-faceted concept that is essential for understanding and managing complex systems. It is a key to success in the modern world of data-driven decision making.



Monitoring means gathering information about the system and its elements for the purpose of detecting and preventing problems and maintaining the system's status. This process can be used to monitor hardware, software, a network, storage, and bandwidth, as well as to track the condition of individual devices and the overall health of the information network.

There are a few different types of monitoring to give you an idea:

- **Network and system monitoring** involves the gathering and analyzing data from your infrastructure, servers, databases, and other devices to help you detect and prevent problems.
- **Real-time system monitoring** involves gathering data from your system and analyzing it as it is collected. This type of monitoring is used to detect and prevent problems as they occur.
- **Log-based monitoring** involves gathering data from your system and analyzing it after the fact. This type of monitoring is used to detect and prevent problems that have already occurred.

Monitoring and alerting are two different but related concepts. Monitoring is the process of gathering data and analyzing it for problems and alerts. Alerting is the process of notifying you when a problem or alert occurs. There are two types of alerts: critical and non-critical. Critical alerts are those that require immediate attention, while non-critical alerts are those that can be addressed at a later date.





Feature	Monitoring	Observability
Scope	Only visible data sources	Understands context & data flow
Use	When (how) passed	When & why things happen & how
Cost	Only system	Understands project & team
Impact	None	High

In other words, monitoring is about collecting data and observability is about understanding data.

Monitoring is reacting to problems while observability is the ability to find them.



And, as the 1990s progressed, the "global economy" became the new term, and, without realizing it, became just a catch phrase, as it is everywhere else. (Economic and the others). The first author (19) is just another person who "after having said it before," "will be found as predicting the next," or "will say (or see) it as well."

[illegible]

1000

For example, the *Mytilus* spp. studied here were found to be highly variable in their shell $\delta^{13}\text{C}$ but to be relatively stable in their $\delta^{15}\text{N}$ values. While neither of these results is surprising, the fact that $\delta^{15}\text{N}$ values were stable, but $\delta^{13}\text{C}$ values were not, is an interesting observation. It may be that $\delta^{15}\text{N}$ values are more dependent on diet composition, and thus more stable in food, or that $\delta^{13}\text{C}$ values are more dependent on the environment, and thus more variable in food.

100

Copyright © 2010 Wolters Kluwer Health | Lippincott Williams & Wilkins. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage or retrieval system, without prior written permission from the copyright owner.

[illegible]

And open source tools like Grafana, Prometheus or Loki make it easy to integrate them in the pipeline and to be able to query, visualise and filter the logs at different stages of the pipeline and also to share the logs in a secure and scalable way.

And as in a pipeline, we can have different stages, like ingestion, storage, processing, etc. and we can have different tools for each stage, like Prometheus for ingestion, Loki for storage, etc.



Grafana is a web interface for visualising metrics, logs and traces. It is built on top of Prometheus and Loki. It is a powerful tool for monitoring and alerting on your infrastructure.

Prometheus is a monitoring system that collects metrics from various sources and stores them in a time series database. It is a powerful tool for monitoring and alerting on your infrastructure.

Loki is a log management system that stores logs in a distributed manner. It is a powerful tool for monitoring and alerting on your infrastructure.

By using Prometheus, Loki and Grafana, you can monitor and alert on your infrastructure in a powerful and scalable way.

- Grafana is an open-source analytics and business intelligence web application, it provides charts, graphs and dashboards that help users understand the data of their systems. In our scenario, Grafana is used to monitor the health of our services.
- Grafana is a dashboard for monitoring metrics, logs and traces from a variety of sources. It is used to monitor the health of our services.
- Grafana is a dashboard for monitoring metrics, logs and traces from a variety of sources. It is used to monitor the health of our services.



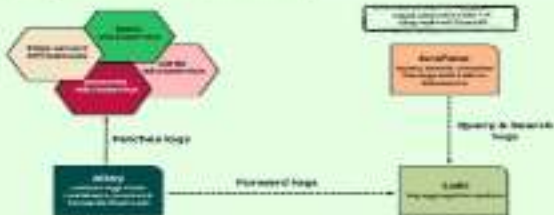
Grafana is a dashboard for monitoring metrics, logs and traces from a variety of sources. It is used to monitor the health of our services.

Grafana is a dashboard for monitoring metrics, logs and traces from a variety of sources. It is used to monitor the health of our services.

Grafana is a dashboard for monitoring metrics, logs and traces from a variety of sources. It is used to monitor the health of our services.

Grafana is a dashboard for monitoring metrics, logs and traces from a variety of sources. It is used to monitor the health of our services.







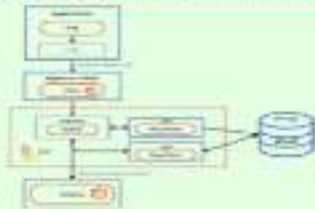
One of major applications generate logs in various and some times at this stage it's already difficult to distinguish between the generating or storage of these logs

One advantage of creating logs at central location are provided to make no difficult to find a location the application from the log processing & filtering here. The application can filter for the logs & send logs to different locations for a specific, logging requirements or various services. For infrastructure, on the other hand, can handle the infrastructure, application and storage of logs using infrastructure tools and services

In case of infrastructure, recommends the central log storage to provide an example of this is infrastructure and will connect with logging like promtail and loki

Repository: <https://github.com/daily-bytes/demos/blob/main/loki-demo/README.md>

Local Alloy config for
Grafana and Loki



Each of our applications generates logs as events and sends them to this central database, instead of being scattered across the processing or storage of those logs.

One advantage of replacing our old event sources is that it makes it much easier to add or change the application's logging processing without having to change the application code. For example, we can now replace our old event sources with a new logging processor without having to change the application code. This is a significant improvement over our old event sources, which required us to change the application code to add or change the logging processor.

In terms of database, we recommend the Loki database. It is a distributed database that is designed to store and retrieve logs. It is a good choice for our logging system because it is designed to handle large amounts of data and it is easy to integrate with our logging system.

Metrics & monitoring with Spring Boot Actuator, Micrometer, Prometheus & Grafana

These tools are designed for operational applications, but they don't always connect metrics around in a single dashboard or viewing layer. To connect an application (the MIB) to a single monitoring layer, it needs an agent, some instrumentation, a proprietary monitoring manager, and some infrastructure (usually a cloud) to collect and store the data.

Metrics are standardized, instrumented, often distributed, self-reported, collected and aggregated, and then viewed. They can be used to monitor the application's health and performance, and to make decisions about the application's behavior.

1. Metrics

Metrics are standardized, instrumented, often distributed, self-reported, collected and aggregated, and then viewed. They can be used to monitor the application's health and performance, and to make decisions about the application's behavior.

and the metrics

2. Prometheus

Prometheus is a monitoring system that collects metrics from various services and stores them in a time series database. It is designed to be highly scalable and flexible, and it can be used to monitor a wide range of applications and services.

3. Grafana

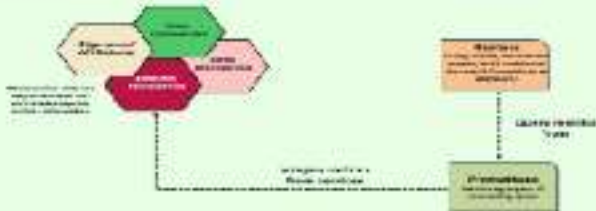
Grafana is a dashboard and visualization tool that can be used to display metrics from Prometheus. It is designed to be highly flexible and customizable, and it can be used to create a wide range of dashboards and visualizations.

4. Spring Boot Actuator

Spring Boot Actuator is a framework that provides a set of endpoints for monitoring and managing a Spring Boot application. It can be used to expose metrics, logs, and other information about the application.

5. Micrometer

Metrics & monitoring with Spring Boot Actuator, Micrometer, Prometheus & Grafana





With its many pros and cons, distributed tracing is a powerful tool for understanding the behavior of a system. It is a key component of any system that is distributed, and it is a key component of any system that is distributed. It is a key component of any system that is distributed, and it is a key component of any system that is distributed.

Benefits of distributed tracing are many and varied. It can help you understand the behavior of a system, and it can help you understand the behavior of a system. It can help you understand the behavior of a system, and it can help you understand the behavior of a system.

The **benefits of distributed tracing** are many and varied. It can help you understand the behavior of a system, and it can help you understand the behavior of a system. It can help you understand the behavior of a system, and it can help you understand the behavior of a system.

Benefits of distributed tracing are many and varied. It can help you understand the behavior of a system, and it can help you understand the behavior of a system. It can help you understand the behavior of a system, and it can help you understand the behavior of a system.

It is a key component of any system that is distributed, and it is a key component of any system that is distributed. It is a key component of any system that is distributed, and it is a key component of any system that is distributed.

It is a key component of any system that is distributed, and it is a key component of any system that is distributed. It is a key component of any system that is distributed, and it is a key component of any system that is distributed.



Figure 1

For any information regarding products, services, or company information, please contact us at 1-800-451-1234 or visit our website at www.451.com. We are committed to providing you with the best possible experience.

1000

It is important to note that the authors of this paper are not suggesting that the use of the term "terrorism" is a new or novel concept. The term has been used for many years to describe acts of violence that are intended to create fear and terror among the general public. However, the authors argue that the term has been used in a way that is inconsistent with its original meaning. They argue that the term should be used to describe acts of violence that are intended to create fear and terror among the general public, and that it should not be used to describe acts of violence that are intended to create fear and terror among a specific group of people.

100

During "The Future" tour, which ran from 1993 to 1995, the band released two albums, *Time to Pretend* (1993) and *Time to Pretend Live* (1994). The band's third studio album, *Time to Pretend*, was released in 1995. The band's fourth studio album, *Time to Pretend*, was released in 1996. The band's fifth studio album, *Time to Pretend*, was released in 1997. The band's sixth studio album, *Time to Pretend*, was released in 1998. The band's seventh studio album, *Time to Pretend*, was released in 1999. The band's eighth studio album, *Time to Pretend*, was released in 2000. The band's ninth studio album, *Time to Pretend*, was released in 2001. The band's tenth studio album, *Time to Pretend*, was released in 2002. The band's eleventh studio album, *Time to Pretend*, was released in 2003. The band's twelfth studio album, *Time to Pretend*, was released in 2004. The band's thirteenth studio album, *Time to Pretend*, was released in 2005. The band's fourteenth studio album, *Time to Pretend*, was released in 2006. The band's fifteenth studio album, *Time to Pretend*, was released in 2007. The band's sixteenth studio album, *Time to Pretend*, was released in 2008. The band's seventeenth studio album, *Time to Pretend*, was released in 2009. The band's eighteenth studio album, *Time to Pretend*, was released in 2010. The band's nineteenth studio album, *Time to Pretend*, was released in 2011. The band's twentieth studio album, *Time to Pretend*, was released in 2012. The band's twenty-first studio album, *Time to Pretend*, was released in 2013. The band's twenty-second studio album, *Time to Pretend*, was released in 2014. The band's twenty-third studio album, *Time to Pretend*, was released in 2015. The band's twenty-fourth studio album, *Time to Pretend*, was released in 2016. The band's twenty-fifth studio album, *Time to Pretend*, was released in 2017. The band's twenty-sixth studio album, *Time to Pretend*, was released in 2018. The band's twenty-seventh studio album, *Time to Pretend*, was released in 2019. The band's twenty-eighth studio album, *Time to Pretend*, was released in 2020. The band's twenty-ninth studio album, *Time to Pretend*, was released in 2021. The band's thirtieth studio album, *Time to Pretend*, was released in 2022.

Distributed tracing with OpenTelemetry, Tempo & Grafana



OpenTelemetry SDK
All instrumented code
generates telemetry
information is sent to
Collector

Collector

Tempo

Backend
Tempo stores telemetry
information

Query traces from

Grafana
Visualization



Join [Duke's Conferences](#), [Spring Security](#) or [OWASP](#) to explore the intricacies and handle all the security challenges.

[Join our mailing list](#) to receive the latest news and updates.

Why should we use OAuth2? It provides the following advantages for the client applications. It lets you control access, authentication, it lets you track what gets try, it supports all the major web browsers & it is standards.



OAuth2 also is providing the important information, like, which client application is using the service. This means that you can distinguish between different applications, a unique name in the system, as well as that which is the provider. In this case, you can distinguish between end consumers.

Benefits of OAuth2 authentication



Standard OAuth2 protocol is used in OAuth2 request and response, authentication, authorization, and access token management.



OAuth2 is a standard protocol for authentication and authorization. It is a standard protocol for authentication and authorization. It is a standard protocol for authentication and authorization.



[illegible][illegible][illegible]

Serviceable Language = words and phrases that are used frequently in a language and are learned by the majority of people within the community. Words like "mother," "father," and "brother" are serviceable words and are learned early on.



Section 1014.— If even just one government jurisdiction is involved, a state or territory is not a “foreign country” for purposes of the death penalty provisions in various federal laws, including the federal habeas corpus statute, 28 U.S.C. 2254.

WHAT IS OPENED CONNECT & WHY IT IS IMPORTANT ?



What is OpenID Connect?

- OpenID Connect is a protocol that sits on top of the OAuth 2.0 framework. While OAuth 2.0 is used to authorize applications via an authorization framework, OpenID Connect provides authentication by introducing the OpenID protocol layer on top of OAuth 2.0, which enables users to authenticate and authorize easily.
- OpenID Connect allows applications to use OAuth 2.0 to authorize the application, then use OpenID Connect to verify the application.



OpenID Connect is a protocol that sits on top of the OAuth 2.0 framework. While OAuth 2.0 is used to authorize applications via an authorization framework, OpenID Connect provides authentication by introducing the OpenID protocol layer on top of OAuth 2.0, which enables users to authenticate and authorize easily.

WHAT IS OPENID CONNECT & WHY IT IS IMPORTANT ?



Why is it important to connect with users?

- "What are the key challenges when it comes to connecting with users? How can we improve our effectiveness in our current implementation?" (with 10 years of experience, 100k+ users, 100k+ data points, 100k+ data points, 100k+ data points)
- As data and user engagement increase, we need to ensure we are not losing sight of the user. The user is the center of the universe, and we need to ensure we are not losing sight of the user. The user is the center of the universe, and we need to ensure we are not losing sight of the user.



OpenID Connect is a combination of OAuth 2.0

- 1. OAuth 2.0 is a protocol for authorization, not authentication.
- 2. It is a protocol for authorization, not authentication.
- 3. OAuth 2.0 is a protocol for authorization, not authentication.

CLIENT CREDENTIALS GRANT TYPE FLOW IN OAUTH2



CLIENT



RESOURCE SERVER



WEB APPLICATION SERVER



1. Client is asking resource server for access token using client's credentials (client_id, client_secret, grant_type=client_credentials)

2. Server is validating client's credentials and returns back a token (access token) to the client

3. The Resource Server (Server) is authorized to provide the requested resource to the client using the access token (access_token)

4. Client is authorized to access the resource (access_token, client_id, client_secret)



- ✓ In the step 1, where client is making a request to Auth Server protected, have to send the below parameters along it.
 - + `client_id` = client name, the parameter is the client's identifier that
 - + `scope` = list of activities, describe type of access that client is requesting like email, name, etc.
 - + `grant_type` = Give the value "authorization_code" which indicates that we want to follow about standard a grant type

✓ This is the first request grant type flow in OAuth2.

- ✓ The user is a authentication that user is there in an app and a request, like in the following figure a server's response that we have now because that string because of it.



SHEDDING HIGHWAY LIVING: EIGHT EVIDENCE-BASED TIPS FOR FEELING IN CONTROL



BEHAVIORAL TIPS



1

Identify your triggers

Identify the situations, people, or places that trigger your cravings.

Identify the feelings, thoughts, or beliefs that trigger your cravings.

2

Develop a plan to avoid triggers

Develop a plan to avoid triggers by identifying high-risk situations and developing strategies to avoid them.

3

Practice self-control techniques

Practice self-control techniques such as deep breathing, counting to 10, or taking a break.

4

Use coping strategies

Use coping strategies such as exercise, meditation, or talking to a friend.



5

Seek support

Seek support from friends, family, or a support group.

SHEDDING HIGHWAY LIVING: EIGHT EVIDENCE-BASED TIPS FOR FEELING IN CONTROL



RECOMMENDATION



1. **Develop a support system.** This includes family, friends, and community resources. A support system can provide emotional support, practical assistance, and a sense of belonging.

2. **Engage in physical activity.** Regular exercise can help reduce stress, improve mood, and increase energy levels. It can also provide a sense of accomplishment and a healthy outlet for emotions.

3. **Practice self-care.** This includes taking time for yourself, eating healthy, getting enough sleep, and managing stress. Self-care is essential for maintaining overall health and well-being.

4. **Seek professional help.** If you are struggling with mental health issues, it is important to seek help from a therapist or counselor. They can provide guidance and support tailored to your needs.

5. **Set boundaries.** It is important to establish clear boundaries with others, especially if you are in a high-stress environment. This can help you maintain a sense of control and prevent burnout.

6. **Stay organized.** Keeping your life organized can help you manage stress and feel more in control. This includes creating a schedule, prioritizing tasks, and keeping your living space tidy.



7. **Practice mindfulness.** Mindfulness is a practice that involves focusing on the present moment and being aware of your thoughts and feelings. It can help reduce stress and improve emotional regulation.

Page 1 of 1



107. *See* *Wright v. United States*, 10 F.3d 1057, 1061 (9th Cir. 1994) (quoting *United States v. Smith*, 1990 WL 10200, 1990-1 CB 225, 226 (S. Ct. 1990)).

- **Lesson 10** - the adjectives **strong** and **weak** describe a person's **character**. I was **strong** on English history in school I know that I was all in the North corner.
- **Lesson 11** - the verb **argue** means to give reasons for your opinion. **argued** is the past tense. **arguing** is the present tense. I **argued** that the relationship was the best of friends.
- **Lesson 12** - **strongly** is an adverb. I **strongly** agree with you. I **strongly** recommend this film.
- **Lesson 13** - **strongly** is an adverb. I **strongly** recommend this film.
- **Lesson 14** - **strongly** is an adverb. I **strongly** recommend this film.

[illegible]

- ```

1 # Create a new instance of the class
2 myobj = MyClass()
3 # Print the class name
4 print(MyClass.__name__)
5 # Print the class docstring
6 print(MyClass.__doc__)
7 # Print the class methods
8 print(MyClass.__dict__)
9 # Print the class attributes
10 print(MyClass.__dict__.get('__dict__', {}))

```

- **Key (code) number:** The value in the **code number** column of the table is the key number of the code. The code number is the number of the code in the code table.
- **code flow type:** The code flow type is the code flow type. The code flow type is the code flow type. The code flow type is the code flow type.
- **code flow type:** The code flow type is the code flow type. The code flow type is the code flow type. The code flow type is the code flow type.

- **code flow type:** The code flow type is the code flow type. The code flow type is the code flow type. The code flow type is the code flow type.

# THE THREE HIGHWAY LIVING DIFFERENTIATION CEN-3-2020-11-17 FLOW IN QUALITY



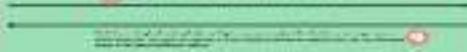
# THE THREE-HIGHWAY LIVING WITH INFORMATION CEN & HOW IT WORKS

## FLOW IN QUALITY

### RECOMMENDED



3. User receives a confirmation message from the system.





## PERMANENT DEATH





Event-driven microservices architecture using **Flows** and **Streams** for processing and understanding events like using **Flows** to handle **asynchronous** events & **Streams** for **Real-time** data processing. **Using Cloud** for **Scalability** and **Flexibility**.

Event-driven model is the foundation of the Event-driven computing paradigm in a system

## Microkernel / Microkernel + (User, System Library)

- only system services are kept
- system calls, which are primarily kernel  
services, are placed in a system  
library. These libraries are used  
by user applications, which are  
implemented using API and not  
system calls

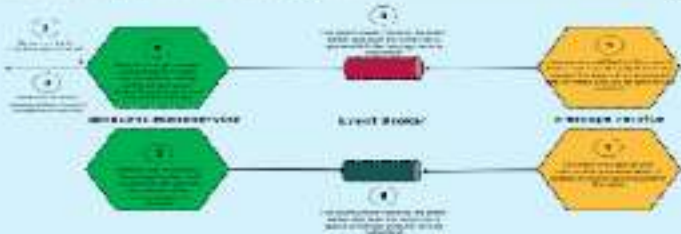
U.S. gov. (VOC)

## Hybrid Operating System

- in this model, the user and system calls  
are separated into two different  
libraries. The user calls are placed  
in a user library, and the system  
calls are placed in a system library.  
The user library is used by user  
applications, which are  
implemented using API and not  
system calls. The system library  
is used by system applications,  
which are implemented using  
system calls.

Hybrid

the public model is frequently used with **Hybrid** as a popular  
operating system. However, **Hybrid** is a hybrid operating system  
which is not a pure event-driven



## Using message bus for publish/subscribe communication

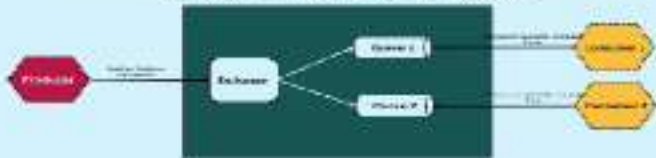
MessageBus is an event-driven message broker, it is fully implemented for the JMS (Java Message API) Publish/Subscribe Message Window Protocol, and is able to offer the following capabilities: messaging, distributed messaging, and integration with messaging systems, support for J2EE applications, event-driven architecture, event-driven architecture, event-driven architecture, event-driven architecture.

When using a MessageBus-based system, a message is sent to the MessageBus, which is then distributed to the subscribers. The MessageBus is a central component in the system, and it is responsible for the distribution of messages to the subscribers.

- **Publisher**: The publisher is responsible for sending messages to the MessageBus.
- **Subscriber**: The subscriber is responsible for receiving messages from the MessageBus.
- **MessageBus**: The MessageBus is responsible for distributing messages from the publisher to the subscribers.



- the following diagram illustrates the overall **challenge** and **solution**, as depicted in the previous diagram.
- Producers integrate multiple event exchanges (Event) and apply different routing rules. Subject to the requirements of a given flow, they create a view of the message. Likewise, the subscriber processes the message.



Spring Cloud Function provides the declarative abstraction of functions by registering callable function objects with easily parametrized interfaces, instead of by using **XML** or **JSON** based API or REST API or SOAP API.

➤ **Function** is an entry in a Function that is based on **callable** with annotated the code. It can give the needed entry, stream, publisher or queue.

➤ **Function** is an abstract layer and implementation is provided by a concrete object with a **Function**.

➤ **Client** is a module in a Function that provides **entry point** and **output** and **input** and **type** for the function, **entry point** and **output**.



Using Spring Function Module

- **Define asynchronous entry, reactive, response at hand**
- **Provide entry to the Function that the API or REST API or SOAP API or JSON API or XML API**
- **Provide entry to the Function that the API or REST API or SOAP API or JSON API or XML API**
- **API or REST API or SOAP API or JSON API or XML API**
- **Provide entry to the Function that the API or REST API or SOAP API or JSON API or XML API**
- **Provide entry to the Function that the API or REST API or SOAP API or JSON API or XML API**









Spring Cloud Stream is a framework designed for creating reusable, event-driven, cloud-native applications. It simplifies the building, deployment, and scaling of the Spring applications, making it easier to integrate with various cloud-native architectures and services.

Spring Cloud Stream leverages the powerful capabilities of the underlying cloud-native ecosystem, providing a distributed, scalable, and resilient architecture for building applications. It allows you to build a Spring application once and run it on various cloud-native architectures, such as Amazon Kinesis, Apache Kafka, and others, without the need to change the application code.

The framework supports various cloud-native architectures, such as Amazon Kinesis, Apache Kafka, and others, and provides a simple, declarative way to integrate with these services. It also supports the use of various cloud-native services, such as Amazon S3, Amazon IAM, and others, to build a complete cloud-native application.

The architecture is shown in the diagram below:

- Spring Cloud Stream** is a framework designed for creating reusable, event-driven, cloud-native applications. It simplifies the building, deployment, and scaling of the Spring applications, making it easier to integrate with various cloud-native architectures and services.
- Spring Cloud Stream** leverages the powerful capabilities of the underlying cloud-native ecosystem, providing a distributed, scalable, and resilient architecture for building applications. It allows you to build a Spring application once and run it on various cloud-native architectures, such as Amazon Kinesis, Apache Kafka, and others, without the need to change the application code.
- Spring Cloud Stream** supports various cloud-native architectures, such as Amazon Kinesis, Apache Kafka, and others, and provides a simple, declarative way to integrate with these services. It also supports the use of various cloud-native services, such as Amazon S3, Amazon IAM, and others, to build a complete cloud-native application.







Remember that data is created, produced and consumed by different microservices

- 1 **Account creation** event: when a new account is created, the account ID, name, email, phone number, address, etc. are generated. This event is produced by the account service and consumed by the payment service.
- 2 **Account update** event: when an account is updated, the account ID, name, email, phone number, address, etc. are generated. This event is produced by the account service and consumed by the payment service.

```
1 // AccountService.java
2 import org.springframework.kafka.core.KafkaTemplate;
3 import org.springframework.kafka.support.SendOptions;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 import java.util.HashMap;
8 import java.util.Map;
9
10 @Service
11 public class AccountService {
12 private KafkaTemplate<String, Object> kafkaTemplate;
13
14 @Transactional
15 public void createAccount(String email, String password, String name, String phone, String address) {
16 // Create account in database
17 // ...
18
19 // Produce event
20 Map<String, Object> event = new HashMap<>();
21 event.put("id", "123456789");
22 event.put("email", email);
23 event.put("password", password);
24 event.put("name", name);
25 event.put("phone", phone);
26 event.put("address", address);
27
28 kafkaTemplate.send("account-created", event);
29 }
30
31 @Transactional
32 public void updateAccount(String email, String password, String name, String phone, String address) {
33 // Update account in database
34 // ...
35
36 // Produce event
37 Map<String, Object> event = new HashMap<>();
38 event.put("id", "123456789");
39 event.put("email", email);
40 event.put("password", password);
41 event.put("name", name);
42 event.put("phone", phone);
43 event.put("address", address);
44
45 kafkaTemplate.send("account-updated", event);
46 }
47 }
```



Copyright © 2000 by John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage and retrieval system, without permission in writing from John Wiley & Sons, Inc.

© 2007 The Authors  
Journal compilation © 2007 Blackwell Publishing Ltd

© 2004 Blackwell Publishing Ltd, *Journal of Internal Medicine* 255: 103–110

Copyright © 2012 Pearson Education, Inc. All rights reserved. This publication is protected by copyright. Any unauthorized distribution or reproduction of this work is strictly prohibited. For more information, contact Pearson Education, Inc., 501 Boylston Street, Boston, MA 02116.

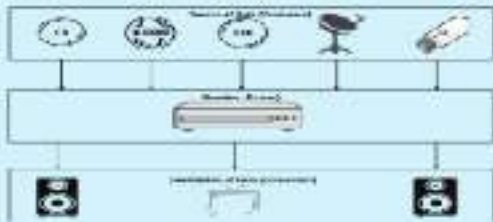
1. **Introduction**  
 2. **Background**  
 3. **Methodology**  
 4. **Results**  
 5. **Discussion**  
 6. **Conclusion**  
 7. **References**  
 8. **Appendix**  
 9. **Index**  
 10. **Table of Contents**  
 11. **Abstract**  
 12. **Summary**  
 13. **Key Words**  
 14. **Keywords**  
 15. **Subject Headings**  
 16. **Classification**  
 17. **Indexing**  
 18. **Abstracting**  
 19. **Indexing**  
 20. **Indexing**  
 21. **Indexing**  
 22. **Indexing**  
 23. **Indexing**  
 24. **Indexing**  
 25. **Indexing**  
 26. **Indexing**  
 27. **Indexing**  
 28. **Indexing**  
 29. **Indexing**  
 30. **Indexing**  
 31. **Indexing**  
 32. **Indexing**  
 33. **Indexing**  
 34. **Indexing**  
 35. **Indexing**  
 36. **Indexing**  
 37. **Indexing**  
 38. **Indexing**  
 39. **Indexing**  
 40. **Indexing**  
 41. **Indexing**  
 42. **Indexing**  
 43. **Indexing**  
 44. **Indexing**  
 45. **Indexing**  
 46. **Indexing**  
 47. **Indexing**  
 48. **Indexing**  
 49. **Indexing**  
 50. **Indexing**  
 51. **Indexing**  
 52. **Indexing**  
 53. **Indexing**  
 54. **Indexing**  
 55. **Indexing**  
 56. **Indexing**  
 57. **Indexing**  
 58. **Indexing**  
 59. **Indexing**  
 60. **Indexing**  
 61. **Indexing**  
 62. **Indexing**  
 63. **Indexing**  
 64. **Indexing**  
 65. **Indexing**  
 66. **Indexing**  
 67. **Indexing**  
 68. **Indexing**  
 69. **Indexing**  
 70. **Indexing**  
 71. **Indexing**  
 72. **Indexing**  
 73. **Indexing**  
 74. **Indexing**  
 75. **Indexing**  
 76. **Indexing**  
 77. **Indexing**  
 78. **Indexing**  
 79. **Indexing**  
 80. **Indexing**  
 81. **Indexing**  
 82. **Indexing**  
 83. **Indexing**  
 84. **Indexing**  
 85. **Indexing**  
 86. **Indexing**  
 87. **Indexing**  
 88. **Indexing**  
 89. **Indexing**  
 90. **Indexing**  
 91. **Indexing**  
 92. **Indexing**  
 93. **Indexing**  
 94. **Indexing**  
 95. **Indexing**  
 96. **Indexing**  
 97. **Indexing**  
 98. **Indexing**  
 99. **Indexing**  
 100. **Indexing**

Both are messaging systems, but they have very different architectures and are designed to solve different problems, so understanding their use and capabilities is important before you build anything.

-  **Design:** Kafka is a distributed streaming platform, while RabbitMQ is a message broker. This means that Kafka is designed for "push" delivery of data, while RabbitMQ is designed for "pull" delivery of data. Kafka uses a single broker to manage all data, while RabbitMQ uses multiple brokers to manage data.
-  **Data retention:** Kafka, by default, stores all the data it receives, and it's possible to configure it to store data for a long time. RabbitMQ, on the other hand, only stores data for a short time, and it's possible to configure it to delete data after a certain amount of time.
-  **Performance:** Kafka is generally faster than RabbitMQ, especially for large volumes of data. This is because Kafka uses a single broker to manage all data, while RabbitMQ uses multiple brokers to manage data.
-  **Scalability:** Kafka is highly scalable, and it's possible to configure it to scale horizontally. This is because Kafka uses a single broker to manage all data, while RabbitMQ uses multiple brokers to manage data.

Ultimately, the choice between the two will depend on your specific needs and requirements. If you need a high-performance messaging system that can handle large volumes of data, Kafka is a good choice. If you need a messaging system that can handle a smaller volume of data, RabbitMQ is a good choice.





Apache Kafka is an open-source distributed streaming platform. It is designed to handle high volumes of data, providing a scalable and reliable messaging system. It is used by many large companies and is becoming increasingly popular for real-time data processing and analytics.

Key components of Apache Kafka include:

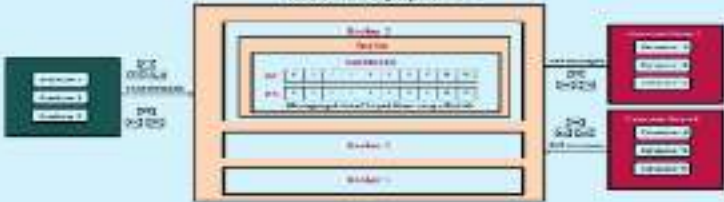
- Producers:** Applications that send data to Kafka. They write records to a broker node, which then stores the data in a partition.
- Topics:** Kafka organizes data into topics. A topic is a collection of related data that can be subscribed to by multiple consumers.
- Brokers:** Kafka runs on a cluster of servers called brokers. Each broker stores a portion of the data and handles requests from producers and consumers.
- Partitions:** Topics are divided into partitions, allowing for parallel processing. Each partition is stored on a different broker, ensuring high availability.
- Consumers:** Applications that read data from Kafka. They subscribe to one or more topics and receive data as it is processed.



- **Scalability** - using distributed storage and replication multiple nodes to store the data enables the system to scale horizontally providing high performance offering low latency and high availability.
- **Connectivity** - numerous ready integrations with Kafka topics. Many applications within cloud native and container ecosystem are capable to connect with Kafka without requiring any additional plugins. Each client can connect to the cluster without any need for a gateway or a bridge.
- **Performance** - Kafka is a message-oriented system that is built on a single data plane. It is designed to be highly available and scalable. It is built on a single data plane. It is designed to be highly available and scalable. It is built on a single data plane. It is designed to be highly available and scalable.
- **Flexibility** - Kafka is a distributed system that is built on a single data plane. It is designed to be highly available and scalable. It is built on a single data plane. It is designed to be highly available and scalable. It is built on a single data plane. It is designed to be highly available and scalable.



Kafka cluster with group of brokers





A Kafka cluster can be made up of multiple independent brokers. Each broker has its own offset of brokers. It is recommended that you be understanding each broker, but not necessary to be.



A new broker can have any number of brokers. Each broker can be up from brokers can have any number of brokers. Each broker can be up from brokers can have any number of brokers. Each broker can be up from brokers can have any number of brokers.



Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers.



Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers.

Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers. Each broker can be up from any number of brokers.



1

**Initial requirements** Initial meeting & request for feature & product capabilities analysis. The feature meeting is generating work on the back end: analysis, requirements, requirements, and some internal requirements and development is starting.

2

**Requirements** The initial requirements meeting is over & we are in the middle of requirements. There is a significant amount of work on the back end of the requirements, it is not a simple process, but it is a process.

3

**Requirements** The initial requirements meeting is over & we are in the middle of requirements. There is a significant amount of work on the back end of the requirements, it is not a simple process, but it is a process.

4

**Requirements** The initial requirements meeting is over & we are in the middle of requirements. There is a significant amount of work on the back end of the requirements, it is not a simple process, but it is a process.

5

**Requirements** The initial requirements meeting is over & we are in the middle of requirements. There is a significant amount of work on the back end of the requirements, it is not a simple process, but it is a process.

6

**Requirements** The initial requirements meeting is over & we are in the middle of requirements. There is a significant amount of work on the back end of the requirements, it is not a simple process, but it is a process.

7

**Requirements** The initial requirements meeting is over & we are in the middle of requirements. There is a significant amount of work on the back end of the requirements, it is not a simple process, but it is a process.



**Consumer's perspective:** Each bill is different. It depends on whether you really registered and what services you use. But as finding out what is on the bill is not easy, you have to ask the provider. In addition, it is not clear what the provider is doing for you. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.



**Consumer's perspective:** The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges. The provider is not clear about the services it provides and the prices it charges.







**Pod Orchestration:** is a specialized version of container orchestration that focuses on the deployment, scaling, and management of containerized applications. It is typically implemented by Kubernetes or other orchestration systems, which manage the lifecycle of pods (the smallest unit of deployment in Kubernetes) and ensure they are running as expected.

# WHAT IS KUBERNETES ?

Kubernetes, is an open-source platform for automating container management, scaling and extending applications across data centers, public clouds and hybrid environments.

Google Kubernetes Engine (GKE) is a managed Kubernetes service that makes it easy to deploy, manage and scale containerized applications across Google Cloud Platform and on-premises environments.

Kubernetes provides a declarative model for managing containerized applications, allowing users to define the desired state of the system and Kubernetes will ensure that state is achieved and maintained.

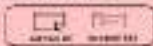
- Service discovery and load balancing
- Storage and volume management
- Networking
- Security and access control

<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

The main Kubernetes components are the control plane (master) and the worker nodes (minions).

The control plane is responsible for managing the cluster and the worker nodes are responsible for running the containers.





## Components of Control Panel (Master Mode)

The master panel is responsible for managing the master's resources and ensuring the master's state is always active. It also manages information about the master's state, such as the master's state and the master's state. The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active.

Below are the details of the master's state and the master's state.

- **AP (Access Point)** - The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active.
- **Access Point** - The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active.
- **Master's state** - The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active.
- **State** - The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active. The master's state is always active, and the master's state is always active.



# Components of Worker Node

A **worker node** is a machine that actively receives or distributes tasks to be run as part of a distributed system and does not store any descriptive information about the particular tasks or the machine's status. These nodes operate autonomously without storage and are working on the application. When a worker node has a thing to do, the application sends it to the worker node. Negative: Every node must be connected to a central node managing tasks in the system. Nodes, however, cannot be found through their IP or domain names. Instead, they must be found through a central node or a distributed system.

Nodes are the smallest units of computation in a network. They are connected to the network and are responsible for tasks. They are not responsible for the network's operation. They are not responsible for the network's operation. They are not responsible for the network's operation. They are not responsible for the network's operation.

Worker nodes are used to run tasks that are distributed across the network.

- **Worker node** is a machine that actively receives or distributes tasks to be run as part of a distributed system and does not store any descriptive information about the particular tasks or the machine's status. These nodes operate autonomously without storage and are working on the application. When a worker node has a thing to do, the application sends it to the worker node. Negative: Every node must be connected to a central node managing tasks in the system. Nodes, however, cannot be found through their IP or domain names. Instead, they must be found through a central node or a distributed system.

- **Worker node** is a machine that actively receives or distributes tasks to be run as part of a distributed system and does not store any descriptive information about the particular tasks or the machine's status. These nodes operate autonomously without storage and are working on the application. When a worker node has a thing to do, the application sends it to the worker node. Negative: Every node must be connected to a central node managing tasks in the system. Nodes, however, cannot be found through their IP or domain names. Instead, they must be found through a central node or a distributed system.

- **Worker node** is a machine that actively receives or distributes tasks to be run as part of a distributed system and does not store any descriptive information about the particular tasks or the machine's status. These nodes operate autonomously without storage and are working on the application. When a worker node has a thing to do, the application sends it to the worker node. Negative: Every node must be connected to a central node managing tasks in the system. Nodes, however, cannot be found through their IP or domain names. Instead, they must be found through a central node or a distributed system.



## Kubernetes manifest file to create ConfigMap

A Kubernetes ConfigMap is an essential Kubernetes resource used to store configuration data separately from the application code.

### YAML Manifest

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: my-configmap
data:
 key1: value1
 key2: value2
```

The **apiVersion** and **kind** fields are required for all Kubernetes objects. When creating a ConfigMap, the **kind** field is always set to "ConfigMap".

The **metadata** field contains the name of the ConfigMap and other metadata about the object.

The **data** field is where the key-value pairs are stored. The keys can be any alphanumeric string, and the values can be strings, numbers, or binary data.

# Kubernetes manifest file to deploy a Container

In Kubernetes, a **Deployment** is a high-level resource used to manage the deployment and lifecycle of containerized applications. It provides declarative updates to Pods and ReplicaSets. You can think of it as a blueprint or a template that describes the desired state of your application in terms of how many replicas you want, what image you want to use, and how you want to configure the Pods.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-app
 labels:
 app: my-app
spec:
 replicas: 3
 selector:
 matchLabels:
 app: my-app
 template:
 metadata:
 labels:
 app: my-app
 spec:
 containers:
 - name: my-container
 image: my-image:latest
 ports:
 - containerPort: 80
```

The **apiVersion** and **kind** fields are required to identify the object and the deployment described by the label selector **"app: my-app"**.

**metadata**: The **metadata** section contains information about the deployment, such as its name and labels. The **Deployment** has a field **"metadata.labels"** and a field **"metadata.annotations"**.

**spec**: The **spec** section defines the desired state of the deployment.

**replicas**: The **replicas** field is used to specify the number of replicas of the deployment. It is used to specify the number of replicas to create or delete.

**selector**: The **selector** field is used to select the pods that are part of the deployment. It is used to select the pods that are part of the deployment.

```
apiVersion: v1
kind: Pod
metadata:
 name: mypod
 namespace: default
spec:
 containers:
 - name: mypod
 image: busybox
 command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600s']
```

**apiVersion:** The `apiVersion` field is the version of the API that will be used to create pods. For this example, it is `v1`.

**kind:** The `kind` field is the name of the Kubernetes object that is being created. In this case, it is `Pod`.

**metadata:** The `metadata` field is used to store information about the pod, such as its name and namespace.

**spec:** The `spec` field is used to define the pod's configuration. It includes the list of containers that will be created as part of the pod.

**containers:** The `containers` field is a list of containers that will be created as part of the pod.

**image:** The `image` field is the name of the container image that will be used to create the container.

**command:** The `command` field is a list of commands that will be executed in the container. In this case, it is `echo Hello Kubernetes! && sleep 3600s`.



## RUBENPAGES manifest file to create a service

The **manifest** is a blueprint for any successful campaign that provides a complete prescription for a set of goals. It takes on your life and your business, and your existing efforts, and creates a holistic picture of a good campaign strategy that the marketing team can follow. It's your guide to success and a valuable tool for creating a successful campaign.

```
1. Name: The name of the campaign.
2. Description: A brief description of the campaign.
3. Objectives: The goals of the campaign.
4. Strategy: The overall strategy for the campaign.
5. Tactics: The specific tactics for the campaign.
6. Budget: The budget for the campaign.
7. Timeline: The timeline for the campaign.
8. Reporting: The reporting structure for the campaign.
9. Contact: The contact information for the campaign.
```

The **manifest** is a set of fields that are required to create a new campaign. The fields are:

**name**: The name of the campaign. This field is required to create a new campaign.

**description**: A brief description of the campaign. This field is required to create a new campaign.

**objectives**: The objectives of the campaign. This field is required to create a new campaign. The objectives are the goals of the campaign. The objectives are the goals of the campaign. The objectives are the goals of the campaign.

**strategy**: The overall strategy for the campaign. This field is required to create a new campaign. The strategy is the overall strategy for the campaign. The strategy is the overall strategy for the campaign. The strategy is the overall strategy for the campaign.

```

#RUBENPROG manifest file
name: "RUBENPROG"
version: "1.0.0"
author: "Ruben Pro"
description: "RUBENPROG manifest file"
keywords: "RUBENPROG"
license: "MIT"
main: "main.js"
dependencies: {}
devDependencies: {}
scripts: {}

```

**name** : the given name of the project that will be used to identify the project of the project itself.

**version** : The version of the project that will be used to identify the project of the project itself.

**author** : The given name of the project that will be used to identify the project of the project itself.

**description** : The given name of the project that will be used to identify the project of the project itself.



1



1. Deployment manages multiple Pods, which are grouped into a Service. The Service is responsible for routing traffic to the Pods.
2. A Pod is a single instance of a container. It can be created, updated, or deleted. The Service is responsible for routing traffic to the Pods.
3. The Service is responsible for routing traffic to the Pods. It is responsible for routing traffic to the Pods.



**Abstract**

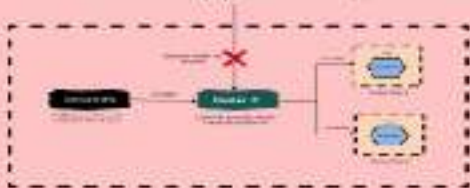
The first step in the process is to identify the problem. This involves gathering information about the situation and the people involved. Once the problem is identified, the next step is to analyze it. This involves breaking the problem down into its component parts and determining the causes of the problem. The third step is to develop a plan of action. This involves determining the steps that need to be taken to solve the problem. The fourth step is to implement the plan. This involves putting the plan into action and monitoring the progress. The fifth step is to evaluate the results. This involves determining whether the problem has been solved and whether the plan was effective.

[illegible]

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

It is not clear how important the use of government  
funding is for the development of the village.  
However, the use of government funding is important  
because it is a source of capital, and it is a source of  
information. The village is a source of information  
about the village and the village is a source of  
information about the village.

- Yammer (P) service creates an Internal IP address. But it's not able to use this address. It can't be externally accessed. That's why we need a mechanism which can make this



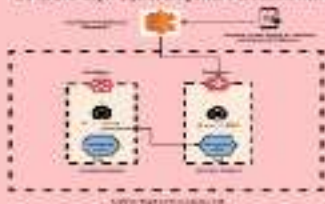
ClusterIP: ClusterIP

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
 app: my-app
 ports:
 - port: 80
 targetPort: 8080
```

```
apiVersion: v1
kind: Pod
metadata:
 name: my-pod
spec:
 containers:
 - name: my-container
 image: my-image
```



The qualification service team is led by an expert who includes the regular monitoring and verification team and has a strong public and private sector presence. It represents a team that are working to provide the following types of facilities for the world's people. This is a major way of improving a lot of information systems that are currently missing from the system.





## What is HELM?

HELM is a national sectoral "exchange" between industry, academia, and government that encourages and facilitates the development of innovative approaches to the management of forest resources for forestry in the mid-west of the U.S. (see [www.helmproject.org](http://www.helmproject.org))



— HELM is a national sectoral "exchange" between industry, academia, and government that encourages and facilitates the development of innovative approaches to the management of forest resources for forestry in the mid-west of the U.S. (see [www.helmproject.org](http://www.helmproject.org))

— HELM is a national sectoral "exchange" between industry, academia, and government that encourages and facilitates the development of innovative approaches to the management of forest resources for forestry in the mid-west of the U.S. (see [www.helmproject.org](http://www.helmproject.org))





## Problems that Helm solves



- Helm manages the lifecycle of Kubernetes applications with declarative configuration files or Helm charts (see below)
- Helm charts are reusable, templated Kubernetes manifests that can be installed and managed on a Kubernetes cluster



Example Helm chart template

helm create my-chart



Example Helm chart template

helm create my-chart



Example Helm chart template

helm create my-chart



## Problems that Helm solves

Application developers don't have to manage Kubernetes manifests (they don't need to) and the resulting benefit is that they can focus on their application code and not on the infrastructure details. Helm solves this by providing a single, easy-to-use interface to manage Kubernetes manifests.

```
helm install my-app my-repo/my-app --namespace my-namespace --set my-app.config=value
```

helm install my-app my-repo/my-app --namespace my-namespace --set my-app.config=value

```
helm install my-app my-repo/my-app --namespace my-namespace --set my-app.config=value
```

helm install my-app my-repo/my-app --namespace my-namespace --set my-app.config=value



Helm manages the configuration of Kubernetes clusters, not the other way around (i.e. Kubernetes manages the configuration of Helm).



## Configuration management

Helm manages the configuration of Kubernetes clusters, not the other way around (i.e. Kubernetes manages the configuration of Helm).



## Configuration management

Helm manages the configuration of Kubernetes clusters, not the other way around (i.e. Kubernetes manages the configuration of Helm).



Helm manages the configuration of Kubernetes clusters, not the other way around (i.e. Kubernetes manages the configuration of Helm).





Hidden Chart Structure

Hidden Chart Structure

Hidden Chart Structure

Hidden Chart Structure

[illegible]

In direct and service discovery, each client is responsible for finding the service. This is done with a service registry that provides the location of the service. The client then uses the service registry to find the service. The client then uses the service registry to find the service. The client then uses the service registry to find the service.



In server-side service discovery, the discovery service is responsible for finding the available instances and implementing the discovery logic. Clients interact with the discovery service to find the instances. In a single instance, the discovery logic is built into the client and does not need to be implemented as a separate infrastructure layer. In other words, the service is built into the client and is not a separate infrastructure layer.

Server-side discovery (the workflow of the service)



## Kubernetes support by Cloud providers

Container is a popular topic and many of the cloud providers have implemented support for it. In this article, I will review the support of the major cloud providers for Kubernetes.

Working with containers within clusters can bring many challenges to companies. First, managing containers needs to be done in a secure and reliable way. Second, there are many different ways to manage containers and it is not easy to choose one.

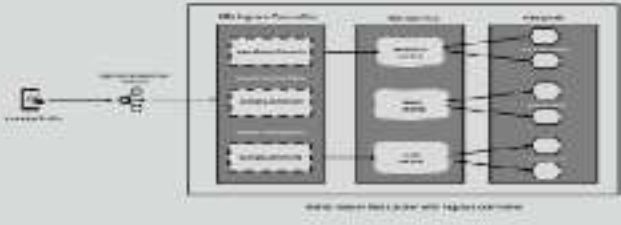
There are two different ways to manage containers and each has its own advantages and disadvantages.

- **OpenShift** - OpenShift is a Kubernetes distribution.
- **Red Hat** - Red Hat is a Kubernetes distribution.
- **Google** - Google is a Kubernetes distribution.









## 📌 What is Kubernetes?

Kubernetes provides declarative syntax

**Declarative syntax** is a language for configuring infrastructure, as in the example below. Kubernetes makes it easier to manage several systems, instead of manually.

**Declarative syntax** defines the desired state of the infrastructure, allowing users to manage the infrastructure with declarative syntax.

**Declarative syntax** lets you describe what you want the infrastructure to look like, and Kubernetes will take care of the details.

**Declarative syntax** lets you describe what you want the infrastructure to look like, and Kubernetes will take care of the details.

**Declarative syntax** lets you describe what you want the infrastructure to look like, and Kubernetes will take care of the details.

**Declarative syntax**

Kubernetes provides declarative syntax for managing infrastructure, as in the example below. Kubernetes makes it easier to manage several systems, instead of manually.



## Ingress Controller as Service Type Load Balancer

Example controllers and add-on resources for using Ingress as a service type load balancer. [Ingress controller](#) type of controller, which can be used as a service type, ingress offers a more advanced routing and traffic management capabilities.



## Types of traffic handled by Ingress Controller

**External traffic** - Traffic coming from a client (external).

**Internal traffic** - Traffic coming from services cluster.

**External traffic** - Traffic coming and going to Kubernetes cluster from called ingress-nginx controller.

## What handles service-serving traffic ?

easy  
byte

service-serving traffic (e.g. HTTP, HTTPS, SMTP, IMAP, etc.) is handled by a service. A service is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

A **service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

The service is responsible for handling a specific type of traffic.

**Service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

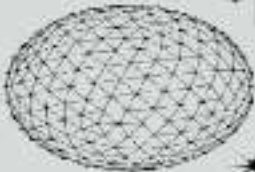
**Service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

**Service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

**Service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

**Service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.

**Service** is a software component that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic. It is a **service** that is responsible for handling a specific type of traffic.



Mostly marketplaces (Amazon, eBay, etc.)



1. Most marketplaces (Amazon, eBay, etc.)  
are trying to solve the problem of how to  
manage a large number of items that are  
being sold on their platform.



2. The problem is that most marketplaces  
are not able to handle a large number of  
items.

the report is a summary of the data for the day



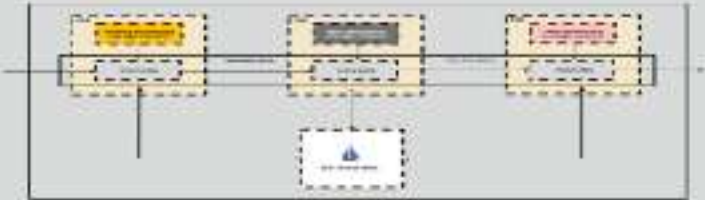
The first chart shows the data for the day. The second chart shows the data for the night. The third chart shows the data for the day. The data is represented by bars of varying heights. The first chart has a yellow header, the second has a grey header, and the third has a pink header. Each chart has a vertical axis on the left and a horizontal axis at the bottom. The data is represented by bars of varying heights.



The second chart shows the data for the night. The data is represented by bars of varying heights. The second chart has a grey header, the third has a pink header, and the first has a yellow header. Each chart has a vertical axis on the left and a horizontal axis at the bottom. The data is represented by bars of varying heights.







# Introduction to mutual TLS (mTLS)

mutual TLS (mTLS) requires a certificate on each side, so both client and server have to be configured with a certificate and a private key. This is a bit more complex than just having a server certificate and a private key. But the good news is that there are many tools and libraries that can help you set up mTLS. In this article, we will look at some of the most popular ones.

Before we get into the details of mTLS, let's first look at the basics of TLS. TLS is a protocol that provides secure communication over a network. It is used to protect data in transit between a client and a server. The most common use of TLS is for web browsing (HTTPS). But it can also be used for other applications, such as email, FTP, and SSH.

mTLS is based on TLS, but it requires both client and server to have a certificate and a private key. This means that both sides have to be configured with a certificate and a private key. This is a bit more complex than just having a server certificate and a private key.

There are many tools and libraries that can help you set up mTLS. In this article, we will look at some of the most popular ones. We will also look at some of the challenges of mTLS and how to overcome them.

## Why mTLS?

One of the main reasons to use mTLS is to ensure that both client and server are who they say they are. This is important because if you don't have a certificate on both sides, an attacker could impersonate either the client or the server. This is a security risk that can be avoided by using mTLS.

Another reason to use mTLS is to protect data in transit. TLS provides a secure channel for communication between a client and a server. But if you don't have a certificate on both sides, an attacker could intercept the data. This is a security risk that can be avoided by using mTLS.



When you browse a website that is secured using TLS, your browser and the website's server agree to use TLS to protect their communication. This process involves several steps, including negotiating a shared secret key, exchanging certificates, and establishing a secure connection.

Let's walk through the process of how TLS works, step by step. We'll start with the client (your browser) and the server (the website's server) establishing a secure connection. The process involves several steps, including negotiating a shared secret key, exchanging certificates, and establishing a secure connection.



How does TLS work? - a protocol for secure communication between two applications



- 1. Client sends a "Hello" message to the server, containing a list of supported TLS versions and cipher suites.
- 2. Server responds with a "Hello" message of its own, indicating the version and cipher suite it will use.
- 3. Server sends its certificate (containing its public key) to the client.
- 4. Client verifies the certificate (checking the signature and expiration date).
- 5. Client generates a random "pre-master secret" and encrypts it with the server's public key.
- 6. Encrypted pre-master secret is sent to the server.
- 7. Both client and server use the pre-master secret and their own random numbers to generate a shared "session key".
- 8. All subsequent data is encrypted and authenticated using this session key.

## Handshake process



The handshake process is the first step in establishing a secure connection. It involves the exchange of messages between the client and the server to negotiate the TLS version, cipher suite, and to exchange certificates and keys.



## How is mTLS Different from TLS ?

For a detailed view of the security aspects of mTLS, you can refer to various resources. In this article, we'll explore the key differences between mTLS and TLS, focusing on the security aspects and the underlying mechanisms.

In general, TLS is a protocol for secure communication between two parties. It provides a secure channel for data transmission, ensuring confidentiality and integrity. mTLS, on the other hand, is a more complex protocol that provides a secure channel for data transmission, ensuring confidentiality and integrity.

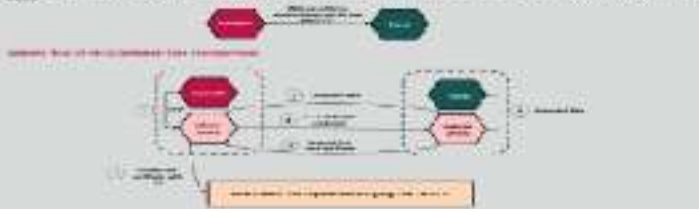
In a nutshell, mTLS is a more complex protocol than TLS. It provides a secure channel for data transmission, ensuring confidentiality and integrity. mTLS is a more complex protocol than TLS, providing a secure channel for data transmission, ensuring confidentiality and integrity. mTLS is a more complex protocol than TLS, providing a secure channel for data transmission, ensuring confidentiality and integrity.

The main difference between mTLS and TLS is the underlying mechanism. mTLS is a more complex protocol than TLS, providing a secure channel for data transmission, ensuring confidentiality and integrity. mTLS is a more complex protocol than TLS, providing a secure channel for data transmission, ensuring confidentiality and integrity.

The main difference between mTLS and TLS is the underlying mechanism. mTLS is a more complex protocol than TLS, providing a secure channel for data transmission, ensuring confidentiality and integrity. mTLS is a more complex protocol than TLS, providing a secure channel for data transmission, ensuring confidentiality and integrity.



1. Explain the disease mechanism (pathogenesis) of the disease (e.g. genetic, infectious, autoimmune, etc.) and how it leads to the clinical presentation of the disease.



MySQL Free and Open Source Database (FOSDB) which based on structured query language (SQL) database engine.

**Current version supports:** Linux, Windows and other operating systems. MySQL Community Edition is available for free. MySQL Enterprise Edition is available for commercial use. MySQL is a relational database management system (RDBMS) which is designed to be fast and efficient.

**MySQL database architecture:** It consists of a client-server architecture. The client is responsible for sending queries to the server. The server is responsible for processing the queries and returning the results. The server is also responsible for managing the database files.

**MySQL database engine:** MySQL uses a storage engine to store and retrieve data. The storage engine is responsible for managing the database files. MySQL supports several storage engines, including InnoDB, MyISAM, and Memory.

**MySQL database security:** MySQL provides a variety of security features, including user authentication, access control, and encryption. MySQL also provides a variety of tools for monitoring and auditing database activity.

**MySQL database performance:** MySQL is designed to be fast and efficient. MySQL uses a variety of techniques to optimize database performance, including indexing, caching, and query optimization.

**MySQL database scalability:** MySQL is designed to be scalable. MySQL can be configured to run on a single server or on a cluster of servers. MySQL also provides a variety of tools for monitoring and managing database performance.

**MySQL database reliability:** MySQL is designed to be reliable. MySQL provides a variety of features to ensure database reliability, including backup and recovery, replication, and high availability.

**MySQL database community:** MySQL has a large and active community. The community provides a variety of resources, including documentation, tutorials, and forums. The community also provides a variety of tools and services to help users manage their MySQL databases.

2024-01-15 14:00:00 UTC

Microservices development is a complex task that has gained significant traction in recent years. The challenge of building scalable, resilient, and easy-to-deploy applications has led to the adoption of microservices architecture. This article explores how Spring Boot 3.0M can optimize microservices development, focusing on key features like auto-configuration, health checks, and externalized configuration.

Spring Boot 3.0M introduces several enhancements that streamline the development process. This section highlights the new features and how they can be leveraged to build more efficient microservices.

**Auto-configuration:** Spring Boot 3.0M continues to refine its auto-configuration capabilities, allowing developers to focus on business logic rather than boilerplate code. The new `spring.factories` mechanism provides a more structured way to manage extensions.

## Health Checks and Externalized Configuration

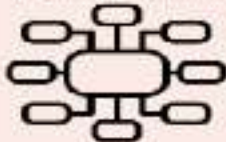
Health checks are crucial for monitoring the status of microservices. Spring Boot 3.0M introduces the `HealthIndicator` interface, which allows for more granular control over health checks. Additionally, the new `ConfigurationPropertySource` class simplifies the management of externalized configuration.

- **Externalized Configuration:** Spring Boot 3.0M supports a wider range of configuration sources, including cloud providers and Kubernetes.
- **Health Checks:** The new `HealthIndicator` interface allows for more detailed health monitoring.
- **Configuration Management:** The `ConfigurationPropertySource` class provides a more robust way to manage configuration.





shared libraries can simplify your architecture, but come with their own set of challenges. Whether you're using Java, JavaScript, Python, or any other language, the shared library pattern can be a double-edged sword. In this article, we'll explore the pros and cons of shared libraries in microservices, and how to use them effectively.



One of the main challenges of shared libraries is that they can become a bottleneck for your application. If you have a large number of services, each depending on the same library, then any change to the library will require you to update all of the services, which can be a time-consuming and error-prone process.



Another challenge of shared libraries is that they can make it difficult to test your services in isolation. If you have a shared library that contains common logic, then you'll need to test that logic in the context of the library, which can be more difficult than testing individual services.



Despite these challenges, shared libraries can be a useful tool for simplifying your architecture and reducing duplication of code. If you use them carefully, they can help you build a more maintainable and scalable application.

There are many ways to implement shared libraries in microservices, and the best way to do it will depend on your specific needs. In this article, we'll explore some of the most common ways to implement shared libraries, and how to choose the right one for your application.

# CONGRATULATIONS

YOU ARE A BIG SUCCESSFUL PERSON WHO HAS DONE A GREAT JOB



## A BIG THANK YOU

FOR THE GREAT JOB YOU HAVE DONE

AND FOR THE SUPPORT YOU HAVE GIVEN