1-

KAFKA	RABBİTMQ	ACTİVEMQ
Mesajı gönderir ancak iletip iletmediği ile ilgilenmez	Mesajın iletilip iletilmediğini kendisine ulaşan acknowledge ile takip eder	Her iletinin teslim durumunu korumak zorundadır
Büyük ölçekli mesajlaşma uygulamalarında kullanılır	Büyük mesajlaşma uygulamaları için uygun değildir	Büyük işletmeleri hedefleyen özelliklere sahiptir
Eski mesajları kolayca çıkarabiliriz	Kalıcı mod ile kullanılmazsa, servis restart edildiğinde bütün mesajlar kaybolacaktır	Sanal bellek, ön bellek kalıcı günlüğü vardır
Tek bir protokol kullanılıyor	Farklı protokoller kullanılabilir	Farklı protokoller kullanılabilir

RabbitMQ için örnek kod parçası verecek olursak;

GÖNDERİM İÇİN;

Gerekli import işlemlerinden sonra sınıfımızı yazıp, sıraya bir isimlendirme yapıyoruz

```
public class Send {
  private final static String QUEUE_NAME = "selam";
  public static void main(String[] argv) throws Exception {
    …
  }
}
```

Sunucu için gerekli bağlantıyı yapıyoruz

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
try (Connection connection = factory.newConnection();
    Channel channel = connection.createChannel()) {
}
```

Akabinde kuyruk bildirimi yapıyoruz

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
String message = "yazacağımız mesaj";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
System.out.println(" [x] Sent '" + message + "'");
```

ALIM İÇİN;

Gerekli import işlemlerinden sonra bağlantı ile birlikte bir kanal açıyoruz ve kuyruğumuzu belirtiyoruz

```
public class Recv {

  private final static String QUEUE_NAME = "selam";

  public static void main(String[] argv) throws Exception {

    ConnectionFactory factory = new ConnectionFactory();

    factory.setHost("localhost");

    Connection connection = factory.newConnection();

    Channel channel = connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false, false, false, null);

    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

  }

}
```

Teslimat için ise;

```
DeliverCallback deliverCallback = (consumerTag, delivery) -> {

    String message = new String(delivery.getBody(), "UTF-8");

    System.out.println(" [x] Received '" + message + "'");

};

channel.basicConsume(QUEUE_NAME, true, deliverCallback, consumerTag -> { });
```

2-

MİCROSERVİS	**MONOLİTH**
Birbirinden bağımsız olarak çalışan ve birbiri ile iletişim kuran servis yapılanmasıdır	UI, Business Logic ve Data Access katmanı tek bir parça halinde tasarlanmıştır yani bütün metodlar tek bir çattı altındadır
Bir yerde yapılan değişiklik diğer yeri etkilemez, değişiklik ilgili yerde yapılacağından daha kolaydır	Bütün bileşenler tek parça altında oluştuğundan bir yerde yapılan çalışma tüm yerleri etkiler
Ekip çalışmasına daha yatkındır çünkü çalışacak olan kişi sadece ilgilendireceği servis ile çalışacaktır	Takım çalışmasında birden fazla kodlama ve yapıda karışıklık meydana gelir
Versiyon yönetimi oldukça kolaydır	Versiyon yönetimi zordur
Herbir servis farklı dilde ve alanlarda yazılabilir	Tek çatı altında oluştuğundan sadece tek platformda ve dilde yazılır

3-

SOAP	**RESTFUL**
XML veri türünü destekler	Daha düşük boyutlarda JSON veri tipi ile işlem yapar
WSDL (web servisleri tanımlama dilli) ile tanımlamada zorunluluk vardır	Zorunluluk yoktur
Geliştirme aracına ihtiyaç duyar	Bunun için ihtiyaç duymaz çünkü tasarlanması kolaydır
XML Scheme kullanır	URI Scheme kullanır
Test ve ayıklama için hata ayıklama aracı gerekir	Herhangi bir ayıklama aracına gerek duymadan http hatalarını döndürür
Http destekler	Http destekler
Cacheleme yapmak için karmaşık XML istekleri gerekir	http get metodunu kullandığı için cacheleme daha kolaydır