Course Preview

Having read about Microservice Architecture, let us now understand how you can deploy those microservices.

In this course, you are going to learn:

* Different patterns and strategies used to deploy services
* Packing services
* Tools and techniques to automate service deployment
* How services communicate among themselves?

**Microservices and Service Deployment**

A Microservice application is made up of tens or even hundreds of services, written in different languages and frameworks.

Each service is a mini-application that must be provided with the appropriate memory, CPU and other resources. In spite of the complexity, deploying services must be reliable, fast and cost-effective.

|  |  |
| --- | --- |
|  | As shown in the graph, the cost of fixing a bug increases exponentially as you move forward in Stages of the Software Life Cycle. Moreover, testing and orchestrating ten's or hundreds of services, manually can be a tedious and buggy experience.  The process should be automated as much as possible from Continuous Integration (CI) to Continuous Deployment (CD). This could save you a lot of time and money. |

**Packaging Services**

Every service might require a different set of dependencies for its execution.

For example: App-1 requires Node-v6 and App-2 requires Node-v8 for its proper execution.

Satisfying dependencies of all the services can be a tedious and challenging job. Hence, you need both the service and its dependencies packed as single Docker or VMI images, before the service deployment.

**Need for Isolation among Services**

You can package all the services as separate Docker or VMI images and create isolation. These images are then used to create the instance of the service. Services isolation is needed for the following reasons:

* Deploying multiple microservices on a VM can influence/disturb other micro-services running on same VM.
* One micro service might generate so many loads and suck all the resources of your machine that the other microservices might die.
* You can easily scale up a Microservice running on individual VM when the load increases.
* When all the processes running on a VM belongs to one micro service, it becomes easy to spot the naughty one to analyze the error.
* You can easily decorate the entire environment of the VM with all the libraries and dependencies required by microservices, and deliver it as a single image (Virtual Machine Image).

|  |  |
| --- | --- |
| **Packaging Services Using Containers** | Packaging Services Using VMI |

**VM vs Container Technology**

Container packages the application first then deploy them on servers. VM is created first on the host machine; then applications are deployed on them.

Both containers and VMs are virtualization technology but they differ in few areas:

* OS: All containers share host machine's OS, while each VMs have their own OS.
* Load: Containers are lightweight, whereas VMs are heavy.
* Security: Containers are less secured, whereas VMs are more secured
* Portability: Docker containers are easily portable but with the same kernel's OS as of previous host machine.

Q:

1. Consider you have an application that does not work correctly in Windows 10 but works fine in Windows XP. Which of the following can be used in your Windows 10 machine to work the application?

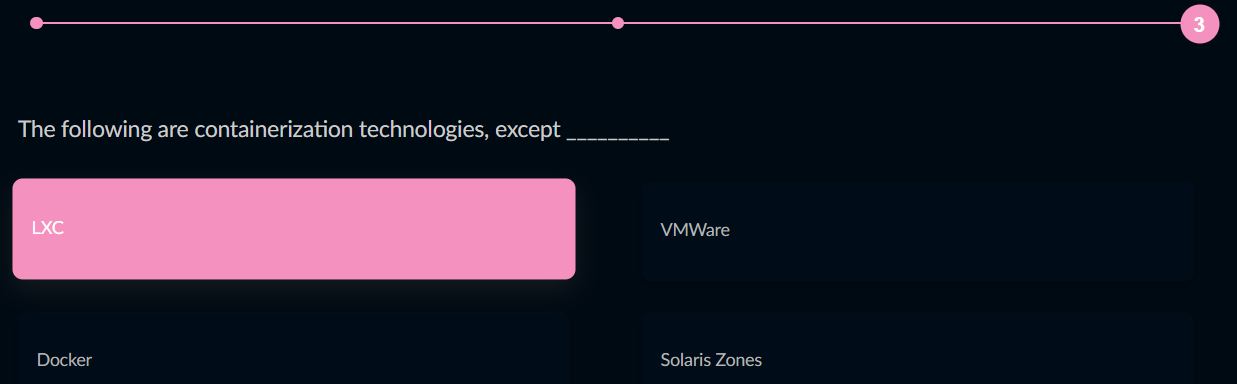
A: 1) Docker or 2) **virtual Machine**

1. You can run multiple OS using \_\_\_\_\_\_\_\_.

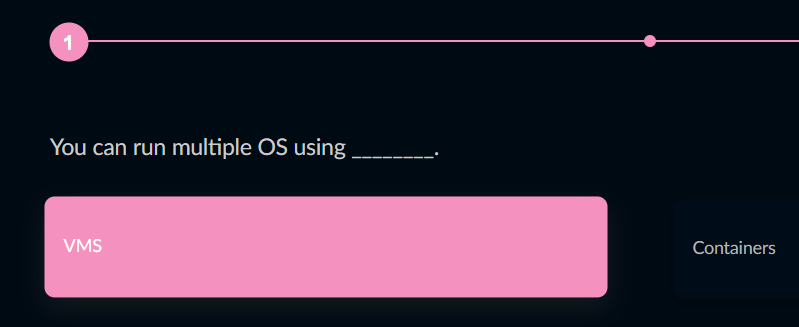
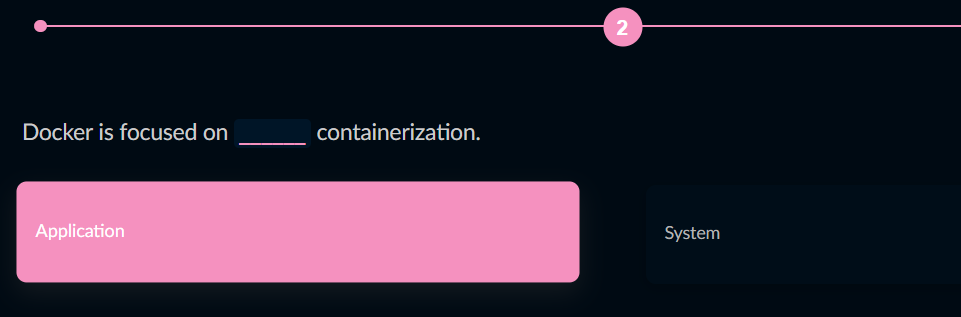
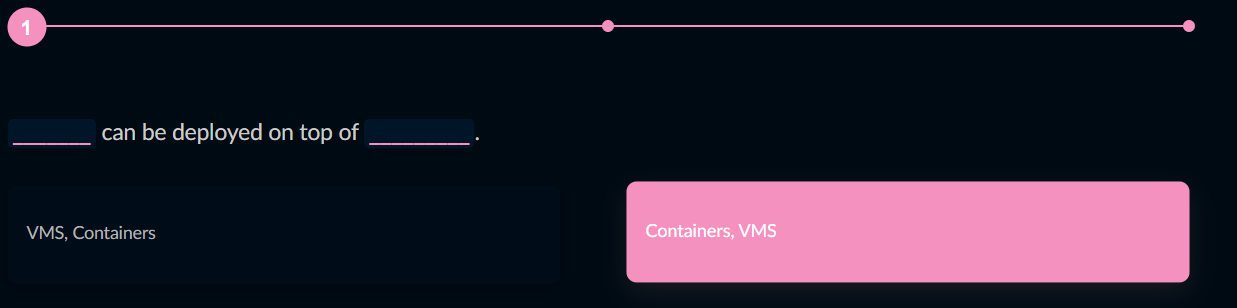
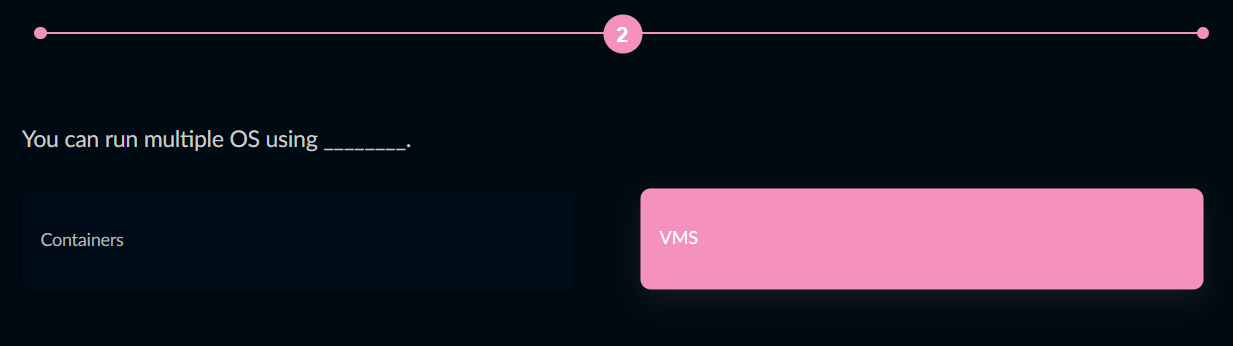
A: 1) Containers 2) **VMS**

1. \_\_\_\_\_\_\_ can be deployed on top of \_\_\_\_\_\_\_\_\_.

A: 1) VMS Containers 2) **Containers VMS**



Correct Ans: VMWare



**Deployment Strategies**

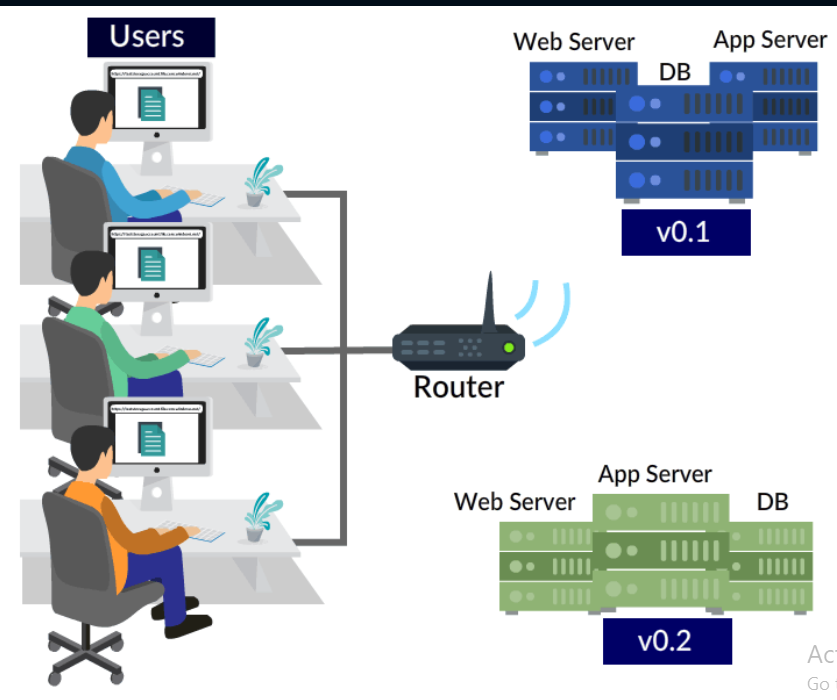
Most common challenges while deploying your services to live production environment, from the final testing stage are:

* to minimize downtime as much as possible and
* To rollback immediately if things did not work out as expected.

You can ensure safer deployments by reducing downtime and risks through following strategies:

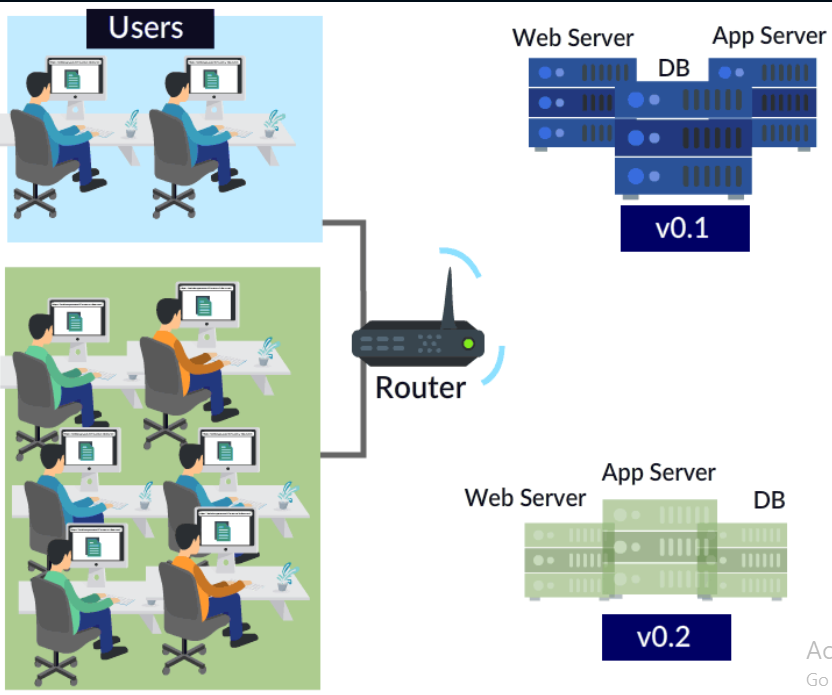
* Blue-Green Deployment
* Canary Releasing

Blue-Green Deployment



Amazon has been practising Blue-Green Deployments since more than 10 years.

##### Canary Release



In Canary Release, you will gradually roll out the new software to a group of users, to verify it is working as expected. Once you are confident with the new version, then you can gradually increase traffic to the new version by deploying it to more servers of your infrastructure.

<https://www.infoq.com/news/2017/09/facebook-release-scale/>

Facebook uses Canary Deployment to achieve rapid release at massive scale.

The spirit of Blue-Green Deployment is deploying at once and the spirit of Canary Deployment is deploying incrementally.

**Same Strategy on Monoliths?**

You might be wondering, why these strategies are implemented with microservices and not monolithic applications?

These strategies are more effective with Microservices, as they

* are much smaller deployment units when compared to monolithic apps
* require less comprehensive tests
* install and start much faster
* Need fewer resources in operation.

**Deployment Patterns**

There are different ways in which you can deploy your microservices:

* Multiple services in single server
* Single service in single server
* Serverless Deployment.

Multiple Services in Single Server

|  |  |
| --- | --- |
|  |  |

**Pros and Cons**

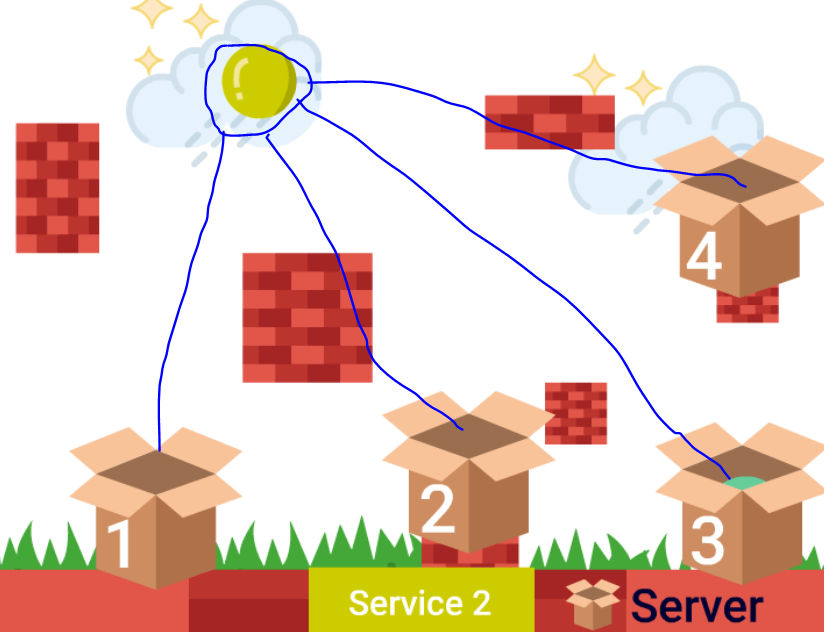
**Pros**:

* Efficient use of resources
* Faster deployments: Just copy the service to host and run it.

**Cons:**

* No isolation of service instance
* You cannot easily monitor or limit resources used by each service instance.
* Complexity increases as microservices can be written in different languages or frameworks, Development team will have to share lots of details (dependencies and libraries to run service) with Operations team to run the service successfully.

Single Service in Single Server



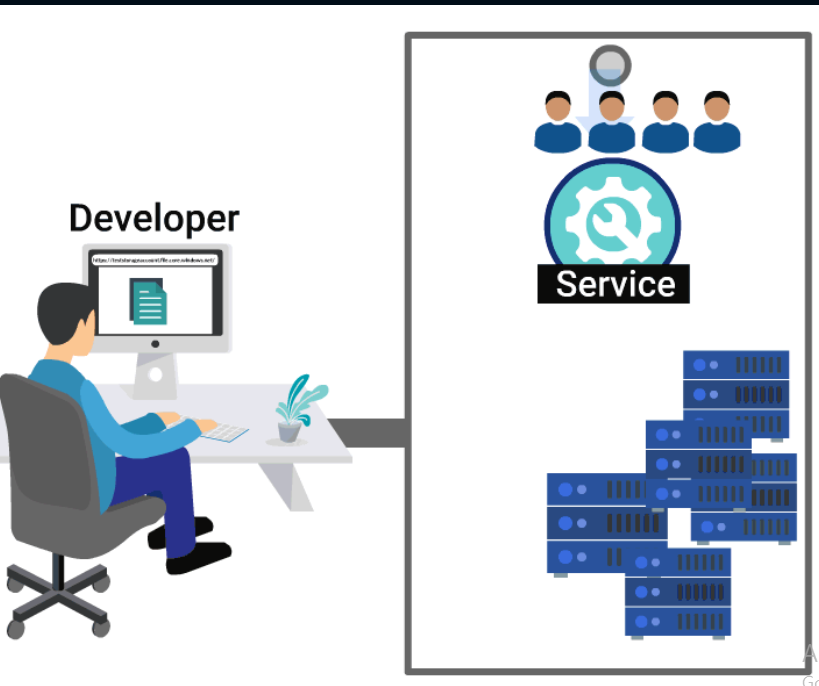
|  |
| --- |
| **Each Service as VMI**  **Pros**:   * Easy to monitor and allocate amount of CPU and memory to each service. * Isolation for each service. * Packing each service as VMI acts as a black box. This helps you to encapsulate the service implementation technology.   **Cons**:   * Less efficient in resource utilization. * VMs are heavy and slow to build (except Boxfuse). |
| **Each Service In a Container**  **Pros**:   * Similar benefits as VM * Lightweight * Fast to build   **Cons**:   * Less mature infrastructure than VM (rapidly increasing though). * Containers share the same kernel of host OS, which makes them less secure than VM. |

**Did You Know?**

Netflix primarily uses **Aminator** tool for packing each service as single VMI (Amazon's EC2 AMI).

<https://netflixtechblog.com/ami-creation-with-aminator-98d627ca37b0>

Serverless Deployment



**Pros and Cons**

**Pros**

* Faster software release.
* Reduced cost of development and operations
* Allows developers to focus on code and deliver updates faster with zero administration work

**Cons**

* Security issue might come as the server and resources are not in your control.
* As you do not control servers, you cannot install any monitoring software. You will have to depend on tools from vendors for monitoring and debugging your services.
* On scaling up, the server might need some time before it could handle requests. This problem is known as cold start.

**Did You Know?**

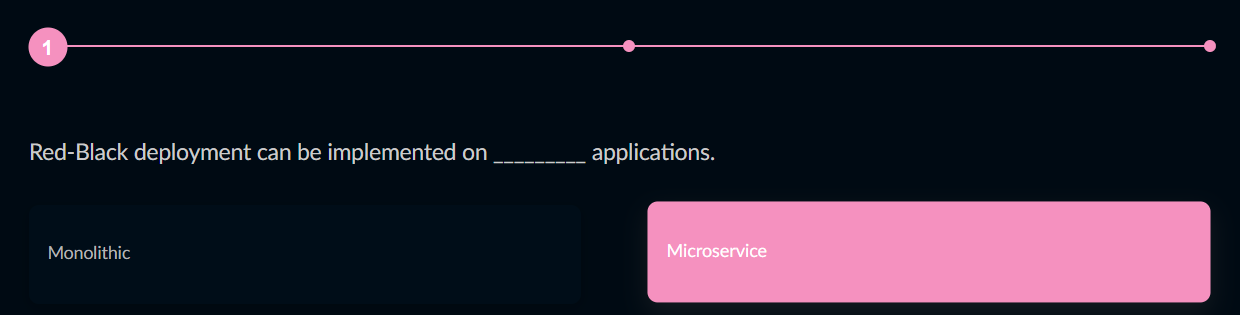
Netflix uses AWS Lambda for managing their AWS infrastructure.

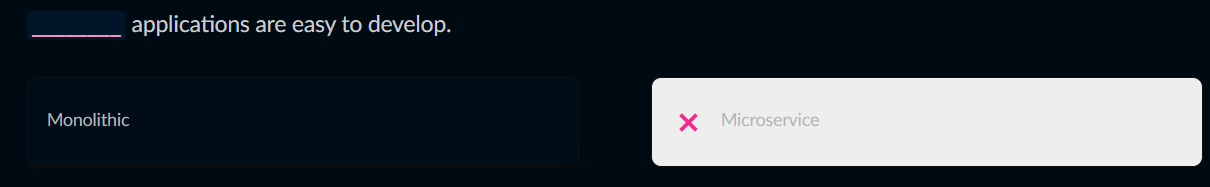
Source: <https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/>

###### **Quiz on Deployment Strategy and Patterns**

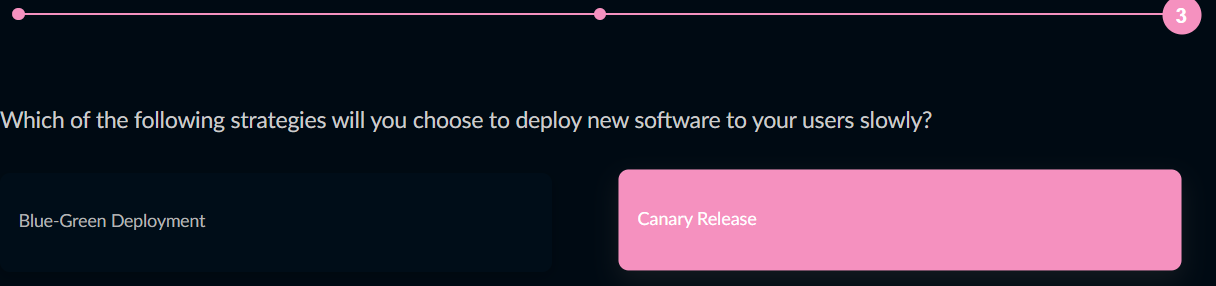
Questions: 3, Passing Score: 2

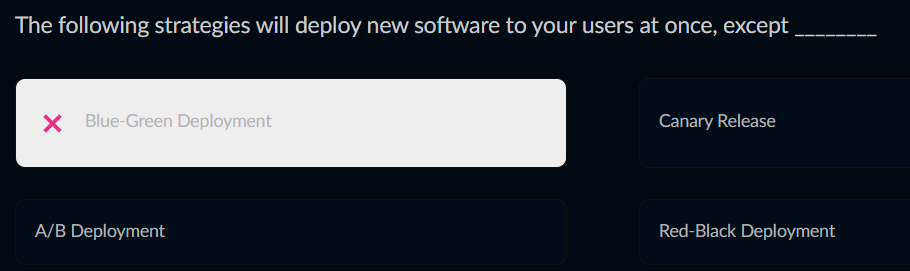
Complete the quiz with minimum score to proceed





Correct Ans: **Monolithic**





**Ways to Automate Deployment**

The main aim of microservices is independent deployment. Manual deployment or correction is not practically possible due to a large number of microservices; the process has to be automated.

There are different ways in which you can automate deploying services and have popcorn enjoying your favourite show. They are

* Installation Scripts
* Deployment Tools

**Installation Scripts**

* Only install the necessary software packages, generate configuration files and create user accounts on your computer
* Such scripts, when called repeatedly, might fail. For example, a script is called to update configuration file or account that is already present in the machine would fail, as they cannot be overwritten easily.
* You can implement these using Shell scripts.

**Deployment Tools**

* You can use other DevOps tools like Puppet, Chef, and Ansible, to deploy and configure your servers.
* You can describe the desired state that your system is supposed to look after installation.
* Running the same installation (for example, Ansible script/playbook) multiple times will not do any further changes to your system as the system is already in the desired state.
* You can easily configure multiple servers at the same time.

**Inter Service Communication**

Just as the phrase goes: Good communication is the key to success in life; so is the communication among your microservices, to make a successfully running application.

Wait; did you just wonder how microservices would communicate when they are in isolation?

Well, you can achieve Service Inter-Operability through:

* Synchronous Communication
* Asynchronous Communication

Asynchronous Communication

**Asynchronous Communication**

In Asynchronous Communication, the client (you may think of your browser) sends a message to a service, assuming it will not receive the reply immediately from service. The client does not get blocked. Hence the user can continue other work.

Example: You can start ten message threads with your ten friends on Fresco Talk and handle the response as they come in (async).

In short, asynchronous communication does not require a response to proceed to the next task.

* Standard protocols used in Asynchronous Communications are AMQP and STOMP.
* Open source messaging systems you can choose from RabbitMQ, Apache Kafka, Apache ActiveMQ, and NSQ.

**Pros and Cons**

**Pros**

* Client and service need not be available at the same time.
* The client need not use service discovery mechanism to determine the location of a service instance.
* No blocking
* Provides good user experience.

**Cons**

* Response time is unpredictable.
* It is more complicated as the client needs to match the response with the request because the service response was not immediate and meanwhile the client would have sent multiple requests to other services.

Synchronous Communication

**Synchronous Communication**

In Synchronous Communication, the client sends a request to service and waits for the service to respond immediately.

Example: In online bank transactions, you are not supposed to refresh website and cannot do any other task on the page until the response comes

In short, the response is a must to proceed to next task in synchronous communication.

* Standard protocols used in Synchronous Communications are REST and Thrift.
* Open-source API designs tools you can choose from RAML and Swagger.

Further Reading: Why Synchronous REST is not recommended for Microservices

<https://thenewstack.io/synchronous-rest-turns-microservices-back-monoliths/>

How Synchronous REST Turns Microservices Back into Monoliths

**Pros and Cons**

**Pros**:

* Simple to implement.
* You can easily test an HTTP API from your browser (using Chrome Extension: Postman) or from CLI (using curl).
* This technique is firewall friendly.
* Response is received immediately.

**Cons**:

* For long-running operations, user experience degrades.
* Client and service should be available for the duration of the exchange.
* Clients must know the location of the service instance, using service discovery.

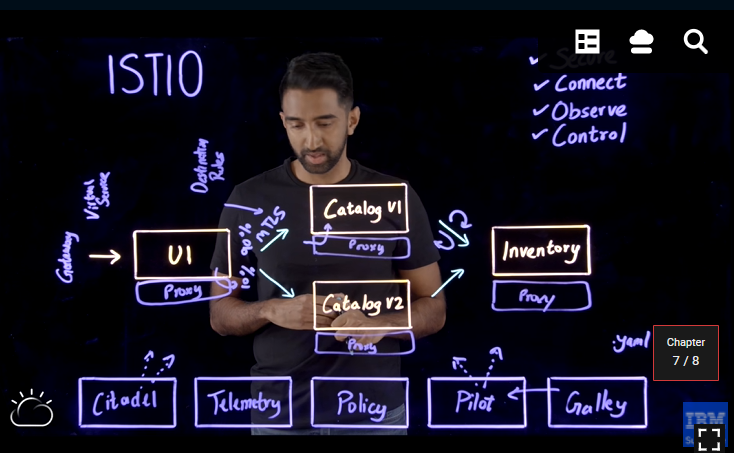
**Message Formats**

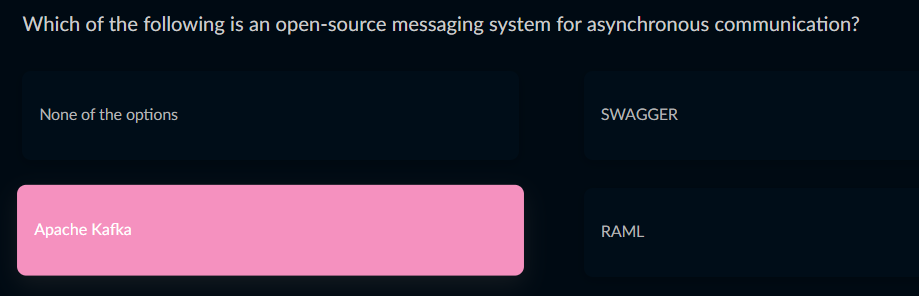
These are two message formats that can be used to transfer data among Microservices.

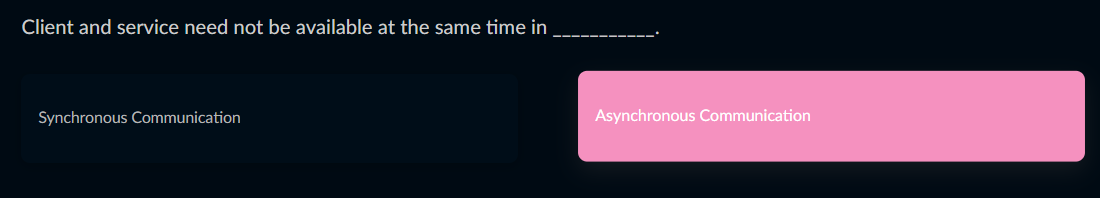
* Text format: JSON, XML
* Binary format

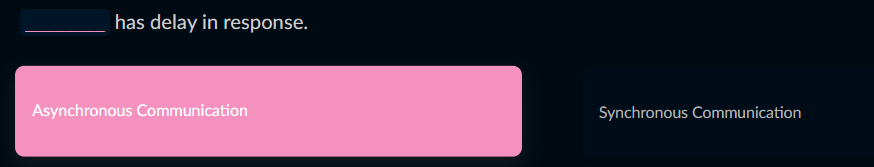
Service Mesh

|  |  |
| --- | --- |
|  | A service mesh is a dedicated infrastructure layer for handling the traffic between service-to-service communications. It is implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware.  **Few tools to implement Service Mesh**: Linkerd and **Istio**. |









**Wrapping Up**

Hope you had fun learning this course.

Let us revise your take away from this course:

The necessity of Automated Deployment.

Packing services and their dependencies in a single box using Containerization or VMI technology.

Different strategies and patterns of micro service deployment.

Scripts you can write to automate deployment.

Different ways and formats for service communication.

Think Tank

Before leaving **a question** to ponder:

Each microservice has its own private place to store data using SQL or NoSQL database.

How can you maintain consistency and retrieve data from multiple services?

**Sayonara!**