Рефлексия. Опыт? Где используется в АСП (Атрибуты, автомапер, конфиги, которые парсятся, DI)

# Amazon SNS Questions:

What is Amazon SNS, and what are its key components? How does it enable communication between distributed systems?

## Explain the difference between a topic and a subscription in Amazon SNS.

* Topic:
  + An endpoint to which messages are published. It acts as a message routing hub.
  + When you publish a message to a topic, SNS delivers that message to all the subscribers (endpoints) that have subscribed to that topic.
  + Topics are essentially the message distribution mechanism, and they decouple the sender (publisher) from the receiver (subscriber) in a publish-subscribe (pub-sub) model.
* Subscription:
  + A subscription represents an endpoint where messages from an SNS topic are delivered. Subscriptions define how messages are delivered and to what destination.
  + Subscribers can be various types of endpoints, such as email addresses, SMS phone numbers, HTTP endpoints, AWS Lambda functions, SQS queues, etc.
  + Each subscription is associated with a specific topic, and when a message is published to that topic, SNS routes the message to all the subscribers' endpoints for that topic.

## Способы доставки сообщений

1. **Standard (Default) Delivery**:
   * **Use Case**: best-effort message delivery without guaranteeing the order of delivery or ensuring that messages are delivered exactly once.
   * **When to Use**: where message loss or duplicate deliveries are acceptable, and the exact order of messages isn't critical. It is cost-effective and works well for notifications, alerts, and broadcasts.
   * **Trade-offs**: Potential for message duplication or loss, and messages might not be delivered in the exact order they were sent.
2. **FIFO (First-In-First-Out) Delivery**:
   * **Use Case**: messages are delivered in the order they were sent and each message is delivered exactly once
   * **When to Use**: Use FIFO when maintaining strict message ordering and eliminating duplicates is essential.
   * **Trade-offs**: FIFO delivery is more expensive than standard delivery, and it has throughput limitations compared to standard SNS topics.
3. **Dead Letter Queue (DLQ) Delivery**:
   * **Use Case**: DLQ delivery is a configuration option rather than a separate delivery mode. It allows you to designate an Amazon SQS queue as a dead letter queue for an SNS topic. Messages that cannot be delivered successfully to subscribers are sent to the DLQ for analysis and troubleshooting.
   * **When to Use**: Use DLQ delivery when you need to capture and investigate undeliverable messages to identify issues with subscribers or message content.
   * **Trade-offs**: DLQs add complexity to your architecture and require monitoring and maintenance to ensure that issues are detected and resolved.
4. **Fanout Delivery**:
   * **Use Case**: Fanout delivery is a pattern where an SNS topic has multiple subscribers, often with different protocols (e.g., email, SMS, HTTP, Lambda, etc.). Messages are sent to all subscribers concurrently.
   * **When to Use**: Use fanout delivery when you want to broadcast messages to multiple subscribers without needing to implement custom logic for message distribution.
   * **Trade-offs**: While fanout delivery is efficient for broadcasting, it can result in a higher cost if you have many subscribers, and it doesn't provide fine-grained control over which subscribers receive specific messages.
5. **Mobile Push Notification Delivery**:
   * **Use Case**: SNS supports sending push notifications to mobile devices (iOS, Android, etc.) using platform-specific protocols (APNS, FCM, etc.). This is commonly used in mobile app push notification scenarios.
   * **When to Use**: Use mobile push notification delivery when you want to send real-time notifications to mobile app users.
   * **Trade-offs**: Requires integration with mobile push notification services (APNS, FCM) and the appropriate setup for each platform.

How does Amazon SNS handle message retries and dead-letter queues? What is the significance of dead-letter queues in SNS?

## Как реализовать фильтрацию сообщений

С поомщью атрибутов

https://medium.com/nuances-of-programming/разветвление-на-различные-очереди-sqs-с-помощью-фильтрации-сообщений-sns-aa83a61d7909

# Amazon SQS Questions:

What is Amazon SQS, and how does it differ from Amazon SNS? When would you choose SQS over SNS for building a messaging system?

Explain the two types of Amazon SQS queues: standard and FIFO (First-In-First-Out). When would you use one over the other, and what are their key characteristics?

How does Amazon SQS ensure message durability and availability? What is the role of message retention periods in SQS?

Discuss the concept of long polling in Amazon SQS. What are the benefits of long polling compared to short polling, and when would you use it?

Can you explain how to handle duplicate messages in Amazon SQS? What strategies or techniques can be employed to avoid processing the same message multiple times?

Describe the use of Amazon SQS dead-letter queues. When and why would you configure a dead-letter queue for an SQS queue, and how does it help in error handling?

Explain how to implement message visibility timeouts in Amazon SQS. What is their role in ensuring message processing reliability?

What is the maximum message size supported by Amazon SQS, and how can you work with larger messages that exceed this limit?

How can you monitor and track the performance and health of Amazon SNS and SQS in a production environment? What AWS services or tools would you use for this purpose?

advantages and disadvantages of synchronous vs asynchronous communication in distributed systems. When would you choose one over the other?

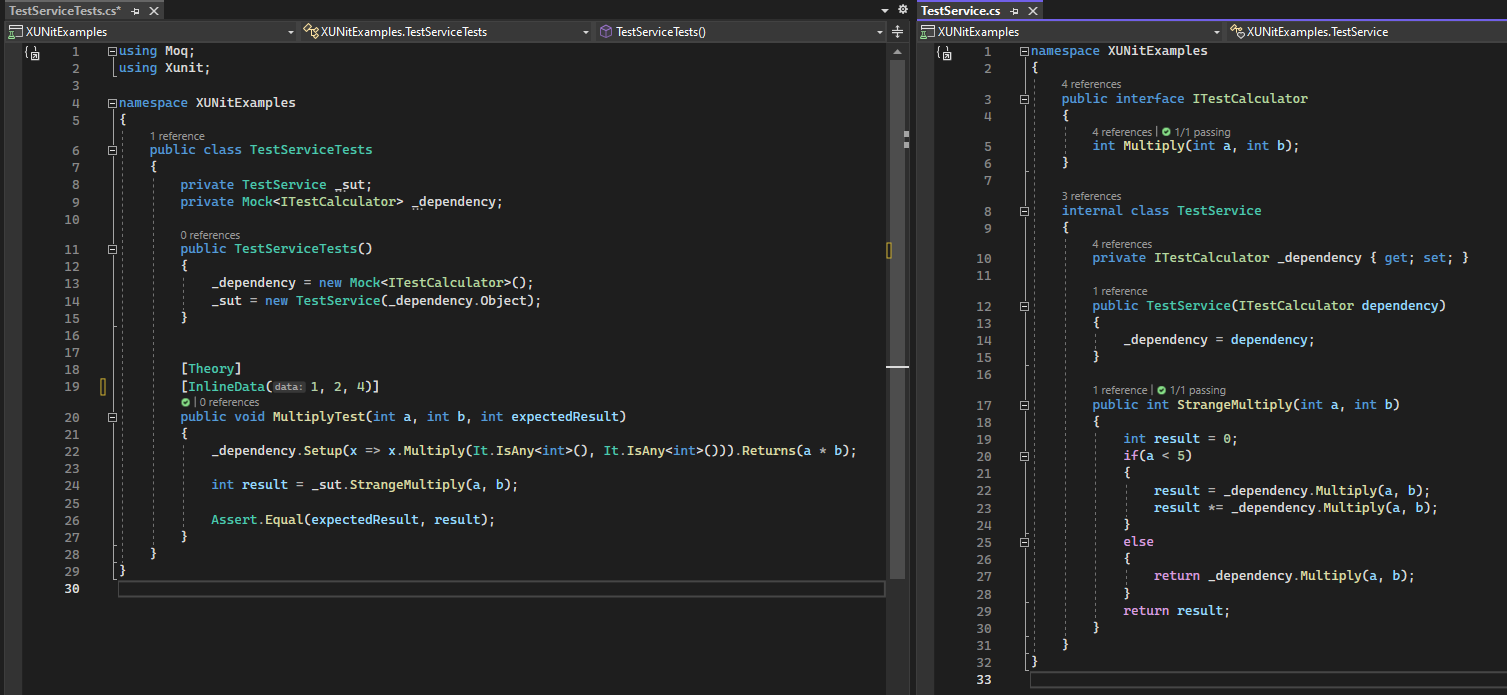
Explain the CAP theorem and its implications in distributed system design. How can you balance consistency, availability, and partition tolerance?

Describe the concept of caching in software architecture. When and why would you use caching, and what caching strategies are available?

# Is that test ok?

What if 1,1,1?

Add \_dependency.Verify to ensure right if branch



Что знаешь помимо солида?

Микросервисы vs монолит vs SOA

[https://aws.amazon.com/ru/compare/the-difference-between-soa-microservices/#:~:text=Микросервисная%20модель%20разделяет%20сервис%20SOA,снижает%20риск%20отказа%20всей%20системы](https://aws.amazon.com/ru/compare/the-difference-between-soa-microservices/#:~:text=%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D0%B0%D1%8F%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%20%D1%80%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%20SOA,%D1%81%D0%BD%D0%B8%D0%B6%D0%B0%D0%B5%D1%82%20%D1%80%D0%B8%D1%81%D0%BA%20%D0%BE%D1%82%D0%BA%D0%B0%D0%B7%D0%B0%20%D0%B2%D1%81%D0%B5%D0%B9%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B).

Горизонтальное vs вертикальное масштабирование, какие инструменты для этого используются? ([Kubernetes](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/),

# What Kubernetes is for?

Kubernetes provides you with:

* **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
* **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
* **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
* **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
* **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
* **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration

What Kubernetes is not

Kubernetes is not a traditional, all-inclusive PaaS (Platform as a Service) system. Since Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions. However, Kubernetes is not monolithic, and these default solutions are optional and pluggable. Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

Kubernetes:

* Does not limit the types of applications supported. Kubernetes aims to support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads. If an application can run in a container, it should run great on Kubernetes.
* Does not deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures and preferences as well as technical requirements.
* Does not provide application-level services, such as middleware (for example, message buses), data-processing frameworks (for example, Spark), databases (for example, MySQL), caches, nor cluster storage systems (for example, Ceph) as built-in services. Such components can run on Kubernetes, and/or can be accessed by applications running on Kubernetes through portable mechanisms, such as the [Open Service Broker](https://openservicebrokerapi.org/).
* Does not dictate logging, monitoring, or alerting solutions. It provides some integrations as proof of concept, and mechanisms to collect and export metrics.
* Does not provide nor mandate a configuration language/system (for example, Jsonnet). It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.
* Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.
* Additionally, Kubernetes is not a mere orchestration system. In fact, it eliminates the need for orchestration. The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C. In contrast, Kubernetes comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state. It shouldn't matter how you get from A to C. Centralized control is also not required. This results in a system that is easier to use and more powerful, robust, resilient, and extensible

# [Service discovery](https://www.baeldung.com/cs/service-discovery-microservices#:~:text=Service%20Discovery%20Implementations&text=The%20client%20queries%20the%20Service,instances%20and%20performs%20a%20request.&text=Giving%20responsibility%20for%20client-side,a%20burden%20and%20an%20advantage.)

Consul

# Стратегии поддержания целостности данных в микросервисной архитектуре

Что знаешь про распределенные транзакции.

САГА vs [двухфазный коммит](https://medium.com/@kirill.sereda/saga-pattern-2daad957966e)

Как повысить отказоустойчивость микросов

Docker, Kubernetes что и зачем?

SQL

join vs union

Как сделать union выборки из двух таблиц с разными столбцами

Select A, B, C, D, E from T1

Union

Select A, B, C, Null as D, Null as E from T2

Union ALL vs Union

Что может возвращать функция в SQL?

Что может возвращать процедура?

Что такое покрывающий индекс?

Фильтрованный индекс? Что? Зачем? Когда?

ACID

Уровни изоляции и проблемы

View / Temp table /Variable table

Global table?

Логгирование

# CI vs CD

1. **Continuous Integration (CI):**
   * **Objective:** CI focuses on automating the process of code integration, testing, and building as frequently as possible. The primary goal is to catch integration issues early in the development cycle.
   * **Process:** Developers commit their code changes to a shared repository multiple times a day. After each commit, an automated build process is triggered, which compiles the code, runs automated tests (unit tests, integration tests, etc.), and performs code analysis.
   * **Benefits:** CI helps identify and address integration issues, bugs, and conflicts between different code branches early in the development process, improving code quality and reducing the time and effort required to fix issues.
2. **Continuous Delivery (CD):**
   * **Objective:** CD extends the principles of CI to ensure that the software is always in a deployable state, ready for release to production at any time. The goal is to automate the deployment and delivery process to minimize manual intervention and reduce the risk associated with releasing new software.
   * **Process:** In CD, the software goes through a series of automated tests, including integration tests, user acceptance tests, and any other necessary checks. If all tests pass, the software is automatically deployed to a staging or pre-production environment, where it undergoes additional testing and validation. If successful, it can then be deployed to the production environment with minimal manual intervention.
   * **Benefits:** CD streamlines the release process, reduces the risk of human error during deployments, and allows for faster and more reliable delivery of software updates to end-users. It also provides the ability to release software features or fixes more frequently.

In summary, CI focuses on the automation of code integration and testing, while CD extends this automation to cover the entire delivery pipeline, from code changes to deployment in production. Together, CI/CD practices help software development teams maintain code quality, improve collaboration, and deliver software more efficiently and reliably.

# Is that test ok?

What if 1,1,1?

Add \_dependency.Verify to ensure right if branch

