**CSCI 391 – ST: Microservices, Quiz 1 – What are Microservices?**

Name:

Student Id:

1. Independent Deployability (a core concept of microservices, and even in its general definition) is the idea that we can make a change to a microservice, deploy it, and release that change to our users, without having to deploy any other microservices.

I’ll also accept “Owning Its Own State” as this is a key factor to enable Independent Deployability.

1. Which of the following (choose 1) best describes what a microservice architecture prioritizes?
2. High cohesion of technical functionality over high cohesion of business functionality.
3. High cohesion of business functionality over high cohesion of technical functionality.
4. True or false, a monolithic architecture is a choice and *a valid one* at that?
5. Which of the following (choose 1) has a database that tends to lack the decomposition we find in the code level, leading to significant challenges if you want to pull apart the monolith in the future?
6. The Distributed Monolith.
7. The Modular Monolith.
8. The Single-Process Monolith.
9. Which of the following (choose 1) has all the disadvantages of a distributed system, *and* the disadvantages of a single-process monolith, without having enough of the upsides of either?
10. The Distributed Monolith.
11. The Modular Monolith.
12. The Single-Process Monolith.

*Extra Credit*

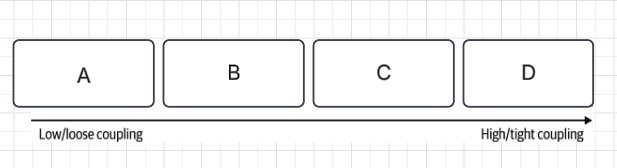
Which company (covered in class) is an example of an organization that has used **The Modular Monolith** as an alternative to microservice decomposition?

Shopify.

**CSCI 391 – ST: Microservices, Quiz 2 – How to Model Microservices.**

Name: **Key**

Student Id: **Key**



*Figure 2-1. The different types of coupling, from loose (low) to tight (high)*

***For questions 1 – 4 identify the correct position; A, B, C, or D (in Figure 2-1) for each type of coupling.***

1. Pass-through

**B**

1. Content

**D (i.e., Pathological Coupling)**

1. Common

**C**

1. Domain

**A**

*Extra Credit: Name a fifth type of coupling (that we covered in class).*

**Temporal Coupling**

***For questions 5 – 8 identify the correct definition for each type of coupling from questions 1 – 4.***

1. Describes a situation in which one microservice needs to interact with another microservice, because the first microservice needs to make use of the functionality that the other microservice provides.

**Domain Coupling**

1. Occurs when two or more microservices make use of a common set of data.

**Common Coupling**

1. Describes a situation in which an upstream service reaches into the internals of a downstream service and changes its internal state.

**Content Coupling**

1. Describes a situation in which one microservice passes data to another microservice purely because the data is needed by some other microservice further downstream.

**Pass-through Coupling**

**CSCI 391 – ST: Microservices, Quiz 3 – Microservices Communication Styles.**

Name:

Student Id:

***For the following scenarios, determine whether they best describe In-process or Inter-process communication.***

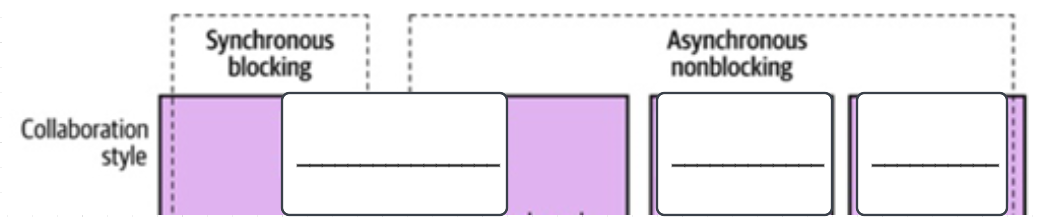
1. Data must be serialized into some form that can be transmitted over a network.
2. In-process
3. Inter-process
4. The act of rolling out changes is straightforward, and in many cases an IDE automatically applies changes for you.
   1. In-process
   2. Inter-process
5. The errors are either expected and easy to handle, or they are catastrophic to the point that we just propagate the error up the call stack. Errors, overall, are deterministic.
6. In-process
7. Inter-process

***For the following statements, determine whether they best describe Synchronous blocking or Asynchronous nonblocking communication.***

1. Its main disadvantage is the inherent temporal coupling that occurs.
   1. Synchronous blocking
   2. Asynchronous nonblocking
2. Its main disadvantage is level of complexity and range of choice.
   1. Synchronous blocking
   2. Asynchronous nonblocking

***For the following scenarios, determine whether Synchronous blocking or Asynchronous nonblocking communication would be the best choice.***

1. Simple microservices architectures.
   1. Synchronous blocking
   2. Asynchronous nonblocking
2. Long running processes.
   1. Synchronous blocking
   2. Asynchronous nonblocking

*Extra Credit (complete the diagram below):* 

Left to right – Request-response, event driven, common data

**CSCI 391 – ST: Microservices, Quiz 4 – Implementing Microservice Communication.**

Name:

Student Id:

1. Which of the following should we consider when selecting a specific technology to implement microservice communication?
   1. Make Backward Compatibility Easy.
   2. Make Your Interface Explicit.
   3. Keep Your API’s Technology Agnostic.
   4. Make Your Service Simple for Consumers.
   5. Hide Internal Implementation Details.
   6. a, b, c, d, and e should all be considered when selecting a specific technology to implement microservice communication.
2. Which of the following technologies can be used to implement synchronous request-response communication?
   1. Remote Procedure Calls.
   2. REST.
   3. GraphQL.
   4. Message Brokers.
   5. a, b, and c can all be used to implement synchronous request-response communication.
3. Which of the following technologies can be used to implement asynchronous request-response communication?
   1. Remote Procedure Calls.
   2. REST.
   3. GraphQL.
   4. Message Brokers.
   5. a, b, and c can all be used to implement asynchronous request-response communication.
4. Which of the following would be used to implement *point to point* asynchronous communication with a single consumer group?
   1. Queue.
   2. Topic.
5. Which of the following would be used to implement asynchronous communication with multiple consumer groups?
   1. Queue.
   2. Topic.

*Extra Credit:*

What is the name of the pattern that enables multiple concurrent consumers to process messages received on the same messaging topic or queue?

Competing Consumers Pattern.

**CSCI 391 – ST: Microservices, Midterm Exam 1**

Name: Key

Student Id: Key

*Success is just around the corner—trust in your preparation and the journey ahead.* **🍀**

**What are Microservices? (16 Points)**

1. Which of the following descriptions best characterizes what the traditional three-tiered architecture (depicted below) prioritizes?



1. High cohesion of technical functionality over high cohesion of business functionality.
2. High cohesion of business functionality over high cohesion of technical functionality.
3. Which of the following are *advantages* of microservices (circle all that apply)?
4. Technology Heterogeneity.
5. Developer Experience.
6. Latency.
7. Ease of Deployment.
8. Scaling.
9. Which of the following are *disadvantages* of microservices (circle all that apply)?
10. Robustness.
11. Cost.
12. Organizational Alignment.
13. Reporting.
14. Monitoring & Troubleshooting.

*For questions 4 to 6, assess whether each scenario represents a good or bad fit for microservices adoption.*

1. A new product or startup where the domain is rapidly evolving and undergoing frequent changes as they refine the core elements of what they are trying to build.
   1. Bad fit for microservices adoption.
   2. Good fit for microservices adoption.
2. A hundred-person scale up that is growing rapidly and wants to allow more people to work independently of each other, reducing delivery contention.
   1. Bad fit for microservices adoption.
   2. Good fit for microservices adoption.
3. An organization creating software that will be deployed and managed by their customers.
   1. Bad fit for microservices adoption.
   2. Good fit for microservices adoption.
4. What are the three different monolithic architectures (where monolith refers to a single unit of deployment)?

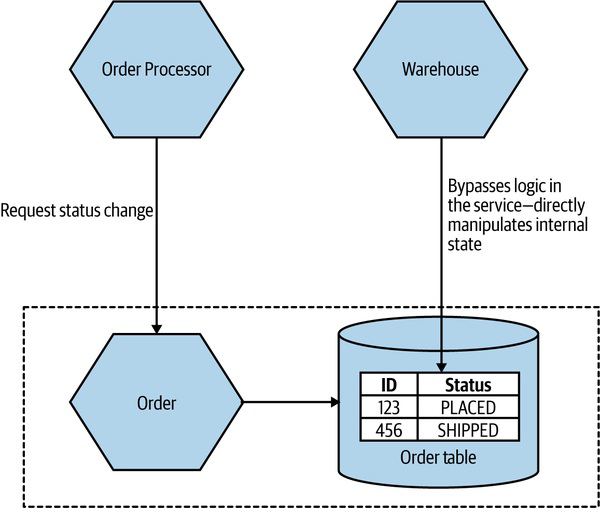
Single process, modular, and distributed monoliths.

1. Which of the following is an advantage of a monolithic architecture?
   1. Simpler deployment topologies.
   2. Simplified code reuse.
   3. Simplified monitoring and troubleshooting.
   4. a, b, and c are all advantages of a monolithic architecture.

**How to Model Microservices (30 Points).**

*For questions 9 to 12, circle the form of coupling that is the lowest, loosest, or most desirable.*

1. Pass-through coupling or Domain coupling.
2. Content coupling or Common coupling.
3. Domain coupling or Common coupling.
4. Pass-through coupling or Content coupling.
5. Which type of coupling is pictured below?



* 1. Domain coupling.
  2. Content coupling.
  3. Common coupling.
  4. Pass-through coupling.

*An order processor microservice makes requests to three different pricing microservices to obtain the best price to order new stock of a specific item. The latency of the response times for the pricing services are as follows:*

|  |  |
| --- | --- |
| **Pricing Microservice** | **Time (ms)** |
| *A* | *1* |
| *B* | *3* |
| *C* | *5* |

1. What is the total latency (duration) of the calls if they happen sequentially?

9 ms

1. What is the total latency (duration) of the calls if they happen parallelly?

5 ms

1. Common coupling can be acceptable in some cases, but not always. Under what conditions might it be acceptable to have common coupling in your system?

Unidirectional flow of data (i.e., one microservice is writing data, one is reading), static reference data.

1. True or false, one aggregate should be managed by more than one microservice?
2. How can we reduce the amount of pass-through coupling and its impact in MusicCorp’s order processing (as described below)? In your response, explain one of the three approaches discussed in the textbook and class, along with potential drawbacks and advantages of the approach you select. Additionally, provide an updated architectural diagram which describes the responsibilities of each microservice in MusicCorp’s updated order process.

*As an example of Pass-through coupling, let’s look more closely now at part of how MusicCorp’s order processing works. In the figure below we have an Order Processor, which is sending a request to Warehouse to prepare an order for dispatch. As part of the request payload, we send along a Shipping Manifest. This Shipping Manifest contains not only the address of the customer but also the shipping type. The Warehouse just passes this manifest on to the downstream Shipping microservice.*



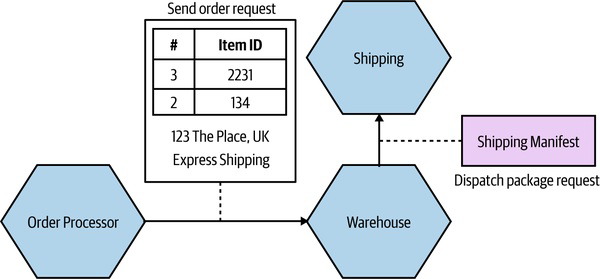
**YOUR ANSWER HERE:**

The major issue with pass-through coupling is that a change to the required data downstream can cause a more significant upstream change. In our example, if Shipping now needs the format or content of the data to be changed, then both Warehouse and Order Processor would likely need to change. There are a few ways this can be fixed. The first is to consider whether it makes sense for the calling microservice to just bypass the intermediary. In our example, this might mean Order Processor speaks directly to Shipping, as in Figure 2-5. However, this causes some other headaches. Our Order Processor is increasing its domain coupling, as Shipping is yet another microservice it needs to know about—if that was the only issue, this might still be fine, as domain coupling is of course a looser form of coupling. This solution gets more complex here, though, as stock must be reserved with Warehouse before we dispatch the package using Shipping, and after the shipping has been done, we need to update the stock accordingly. This pushes more complexity and logic into Order Processor that was previously hidden inside Warehouse.



*Figure 2-5. One way to work around pass-through coupling involves communicating directly with the downstream service.*

For this specific example, I might consider a simpler (albeit more nuanced) change—namely, to totally hide the requirement for a Shipping Manifest from Order Processor. The idea of delegating the work of both managing stock and arranging for dispatch of the package to our Warehouse service makes sense, but we don’t like the fact that we have leaked some lower-level implementation— namely, the fact that the Shipping microservice wants a Shipping Manifest. One way to hide this detail would be to have Warehouse take in the required information as part of its contract, and then have it construct the Shipping Manifest locally, as we see in Figure 2-6. This means that if the Shipping service changes its service contract, this change will be invisible from the viewpoint of Order Processor, as long as Warehouse collects the required data.



*Figure 2-6. Hiding the need for a* Shipping Manifest *from the* Order Processor.

While this will help protect the Warehouse microservice from some changes to Shipping, there are some things that would still require all parties to change. Let’s consider the idea that we want to start shipping internationally. As part of this, the Shipping service needs a Customs Declaration to be included in the Shipping Manifest. If this is an optional parameter, then we could deploy a new version of the Shipping microservice without issue. If this is a required

parameter, however, then Warehouse would need to create one. It might be able to do this with existing information that it has (or is given), or it might require that additional information be passed to it by the Order Processor.

Although in this case we haven’t eliminated the need for changes to be made across all three microservices, we have been given much more power over when and how these changes could be made. If we had the tight (pass-through) coupling of the initial example, adding this new Customs Declaration might require a lockstep rollout of all three microservices. At least by hiding this detail we could much more easily phase deployment. One final approach that could help reduce the pass-through coupling would be for the Order Processor to still send the Shipping Manifest to the Shipping microservice via the Warehouse, but to have the Warehouse be totally unaware of the structure of the Shipping Manifest itself. The Order Processor sends the manifest as part of the order request, but the Warehouse makes no attempt to look at or process the field—it just treats it like a blob of data and doesn’t care about the contents. Instead, it just sends it along. A change in the format of the Shipping Manifest would still require a change to both the Order Processor and the Shipping microservice, but as the Warehouse doesn’t care about what is in the manifest, it doesn’t need to change.

*For questions 19 to 21, choose the strategy described in each scenario as an alternative to using the Business Domain as the boundary for microservice modeling.*

1. Identify the parts of the system that experience more frequent changes, and extract those functionalities into separate services, allowing for more effective development and maintenance.
   1. Technology.
   2. Data.
   3. Volatility.
   4. Organizational.
2. Services within the green zone (outlined by a dotted green line) never have access to credit card information. This sensitive information is restricted to processes and networks within the red zone (outlined by red dashes). The gateway directs calls to the appropriate services and zones, ensuring that when credit card information passes through the gateway, it is treated as being within the red zone.
   1. Technology.
   2. Data.
   3. Volatility.
   4. Organizational.
3. Some of your functionality requires implementation in a runtime like Rust, which allows you to achieve higher performance gains.
   1. Technology.
   2. Data.
   3. Volatility.
   4. Organizational.

**Microservices Communication Styles (20 Points).**

*For questions 22 to 24, determine whether each scenario best describes In-process or Inter-process communication.*

1. The underlying compiler and runtime can carry out a whole host of optimizations to reduce the impact of the call, including inlining the invocation so it’s as though there was never a call in the first place.
   1. In-process.
   2. Inter-process.
2. When making a backward-incompatible change to an interface, you either need to do a lockstep deployment with consumers, making sure they are updated to use the new interface, or else find some way to phase the rollout of the new contract.
   1. In-process.
   2. Inter-process.
3. You are vulnerable to a host of errors that are outside of your control i.e., network timeouts.
   1. In-process.
   2. Inter-process.
4. What is the difference between 400 and 500 series response codes (which are both reserved for errors) in the HTTP protocol?

400 – something is wrong with the request. 500 – something is wrong with the downstream microservice.

*For questions 26 to 28, determine whether each scenario best describes Synchronous blocking or Asynchronous nonblocking communication.*

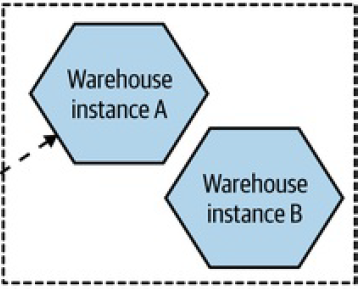
1. Becomes problematic when you start having more chains of calls.
   1. Synchronous blocking.
   2. Asynchronous nonblocking.
2. Every call would also be considered a request-response call.
   1. Synchronous blocking.
   2. Asynchronous nonblocking.
3. The microservice making the initial call and the microservice (or microservices) receiving the call are decoupled temporally.
   1. Synchronous blocking.
   2. Asynchronous nonblocking.

*For questions 29 to 31, determine the best Asynchronous nonblocking communication style to implement based upon the given scenario.*

1. There is a need for interoperability between different types of systems, including older mainframe applications or customizable off-the-shelf (COTS) software products.
   1. Common data.
   2. Event-driven.
   3. Request-response.
2. An upstream microservice need to make sure something gets done by a downstream microservice, and in fact asks it to.
   1. Common data.
   2. Event-driven.
   3. Request-response.
3. A service broadcasts information and assumes that interested parties will react accordingly.
   1. Common data.
   2. Event-driven.
   3. Request-response.

**Implementing Microservice Communication (10 Points).**

1. These serialization protocols are what you want to use if you start getting worried about payload size or about the efficiencies of writing and reading the payloads.
   1. Textual formats.
   2. Binary formats.
2. One or many instances of the same microservice (pictured below) that work together to process messages from a message broker, such as Kafka, RabbitMQ, or others is referred to as what?



Consumer Group

1. Which of the following would be used to implement point to point asynchronous communication between two microservices?
   1. Topic.
   2. Queue.
2. The OpenAPI specification (which originated from the Swagger project) now provides the ability to define enough information about endpoints to generate client-side code in various languages for which technology?
   1. Remote Procedure Calls.
   2. REST.
   3. GraphQL.
   4. Message Brokers.
3. Which of the following technology is also considered a *middleware*?
   1. Remote Procedure Calls.
   2. REST.
   3. GraphQL.
   4. Message Brokers.

**CSCI 391 – ST: Microservices, Quiz 5 – Implementing Microservice Communication.**

Name:

Student Id:

*For questions 1 & 2, select the type of contract breakage that is possible for each scenario.*

1. Fields or methods being removed, or new required fields being added to a microservice endpoint.
   1. Semantic Breakage.
   2. Structural Breakage.
2. The behavior of a microservice endpoint changes in such a way as to break consumers’ expectations.
   1. Semantic Breakage.
   2. Structural Breakage.
3. What key idea helps avoid breaking changes by implementing a client (reader) that can ignore changes that are not relevant?

Tolerant Reader.

1. If a piece of software uses semantic versioning and its version is incremented from 1.0.0 to 1.1.0 in the next release, what changes can I infer have occurred based on the rules of semantic versioning?
   1. New functionality has been added that should be backwards compatible.
   2. Backwards incompatible changes have been made.
   3. Bug fixes have been made to existing functionality.
2. Which option for managing the rollout of breaking changes to a microservice contradicts the principle of independent deployability, as it requires both the microservice and all of its consumers to be updated simultaneously?
   1. Emulate the old interface.
   2. Coexist incompatible microservice versions.
   3. Lockstep deployment.

*Extra Credit:*

Due to the TTL (Time to Live) of DNS records and their frequent caching in multiple locations, which technology can help prevent referencing outdated IP addresses of a microservice instance, and allow us to refer to this technology instead of individual microservice IPs in our DNS records?

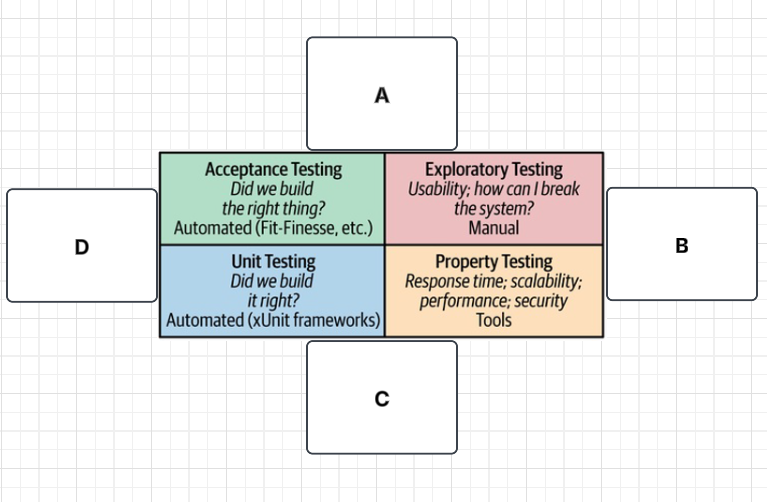
Load Balancer.

**CSCI 391 – ST: Microservices, Quiz 6 – Testing.**

Name:

Student Id:

*For questions 1 – 4 (A – D) complete Marick’s quadrant (below) for “types of testing” i.e., please select one of the following for* ***A****,* ***B****,* ***C****, and* ***D (1-4)****;* ***Business facing, Technology facing, Support programming,*** *or* ***Critique product.***



1. (**A**)

Business facing

1. (**B**)

Critique product

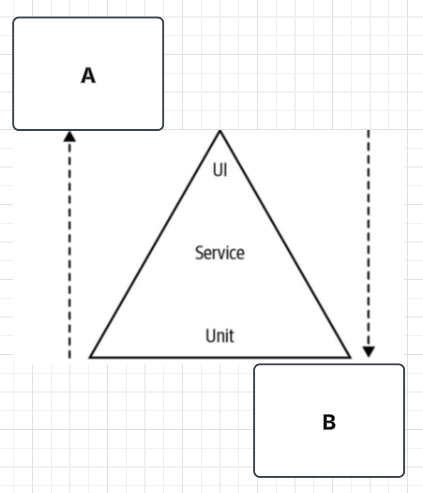
1. (**C**)

Technology facing

1. (**D**)

Support programming

*For questions 5 – 8 Please complete the test pyramid (below) for “scope of testing” i.e., please match the following to either position* ***A*** *or* ***B*** *on the test pyramid…*



1. Increasing Scope.

A

1. Faster.

B

1. Better isolation.

B

1. More confidence.

A

*Extra Credit:*

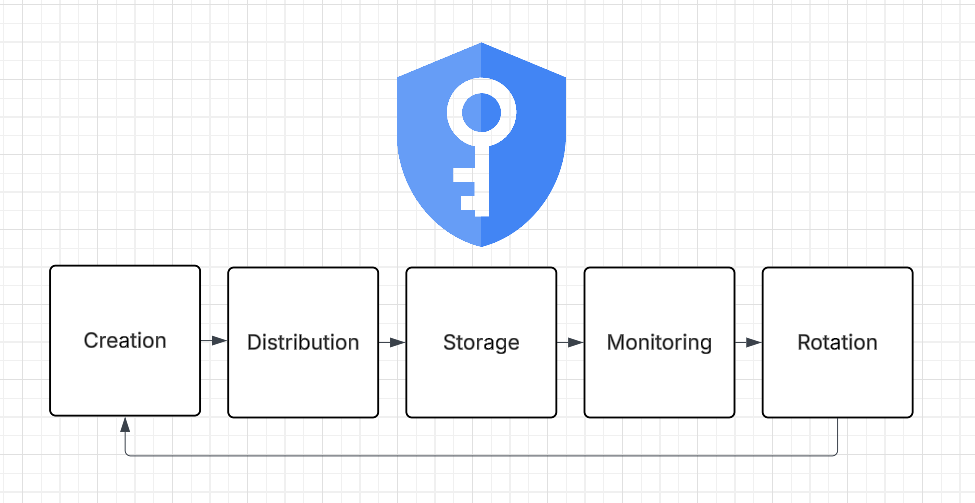
A common antipattern where the overall distribution of your tests reflects the inverted test pyramid. Test snow cone antipattern.

**CSCI 391 – ST: Microservices, Quiz 7 – Security.**

Name:

Student Id:

1. Which of the following is **NOT** one of the five core functions of cybersecurity?
   1. Identify
   2. Secure
   3. Protect
   4. Respond



*For questions 2 & 3 identify the step in the “lifecycle of a secret” (above) described…*

1. Knowing how the secret is being used.

Monitoring

1. Making sure the secret gets to the right place (and only the right place).

Distribution

*For questions 4 & 5 decide whether the scenario describes implicit trust or zero trust…*

1. Assume that any calls to a service within our security perimeter are trusted.

Implicit Trust

1. Assume that the environment is already compromised that is being operated in.

Zero Trust

*Extra Credit:*

This is a backup that may or may not actually be a backup (give the name).

Schrodinger Backup

**CSCI 391 – ST: Microservices, Midterm Exam 2**

Name:

Student Id:

"You shall not pass... without proper testing! 🧙‍♂️🛡️"

**Implementing Microservice Communication (20 Points).**

1. MusicCorp’s customer profile team updated the **create customer** endpoint by renaming the "name" field to "fullName" in the request body. This change to the schema makes previous versions incompatible, requiring clients to update their code to use the new field….

**Before the Change:**

{

"name": "Addison Smith",

"email": "addison.smith@example.com"

}

**After the Change:**

{

"fullName": "Addison Smith",

"email": "addison.smith@example.com"

}

1. Identify the type of contract breakage described above (choose one):
2. Structural Breakage.
3. Semantic Breakage.
4. Select the design principle that was not followed by MusicCorp’s customer profile team, which if followed, could have prevented the breakage (choose one):
5. Expansion Changes.
6. Tolerant Reader.
7. The breakage could not have been prevented.
8. Assuming MusicCorp’s customer profile team had followed the design principle from part B, provide an example of what the updated request contract should look like (in JSON format).

{“name”: “Addison Smith”, “fullName”: “”, “email”: “a@a.com”}

1. MusicCorp's customer profile team updated the **get customer** endpoint, which returns customer details like name, email, and address in the response body. The **fullName** field now includes the first name, middle initial, and last name (e.g., "Jamie L Smith"), whereas it previously included only the first and last name (e.g., "Jamie Smith"). Although the schema hasn't changed, consumers expecting the old format may encounter issues, as the updated behavior breaks their expectations….

**Before the Change:**

{

"customerId": "12345",

"fullName": "Jamie Smith",

"email": "jamie.smith@example.com",

"address": "123 Main St, New York, NY"

}

**After the Change:**

{

"customerId": "12345",

"fullName": "Jamie L Smith",

"email": "jamie.smith@example.com",

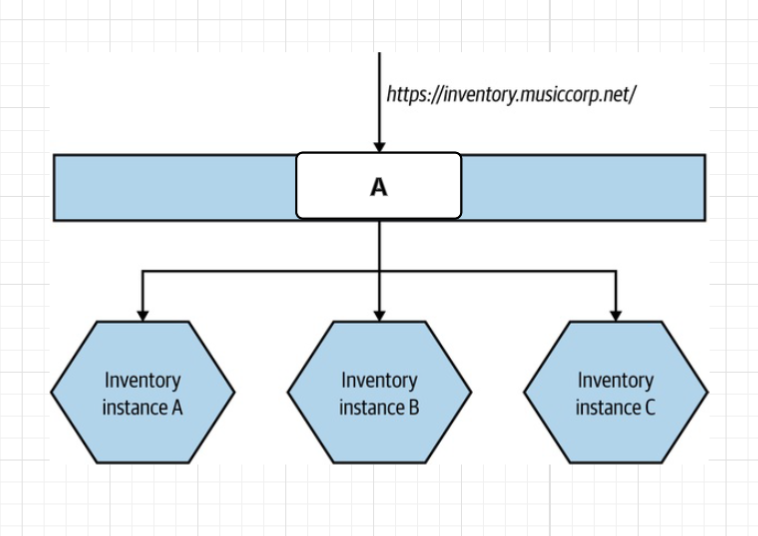
"address": "123 Main St, New York, NY"

}

* 1. Identify the type of contract breakage described above (choose one):

1. Structural Breakage.
2. Semantic Breakage.
   1. Select the design principle that was not followed by MusicCorp’s consumers, which if followed, could have prevented the breakage (choose one):
3. Expansion Changes.
4. Tolerant Reader.
5. The breakage could not have been prevented.
6. A microservice currently uses *semantic versioning*. If its version is incremented from ***1.0.0*** *to* ***1.0.1***in the next release, what type of change does this indicate according to semantic versioning rules (choose one)?
7. New functionality has been added that should be backwards compatible.
8. Backwards incompatible changes have been made.
9. Bug fixes have been made to existing functionality.
10. Which of the following options for dealing with breaking changes is the least desirable because it compromises the independent deployability of a service (choose one)?
11. Coexist incompatible microservice versions.
12. Lockstep deployment.
13. Emulate the old interface.

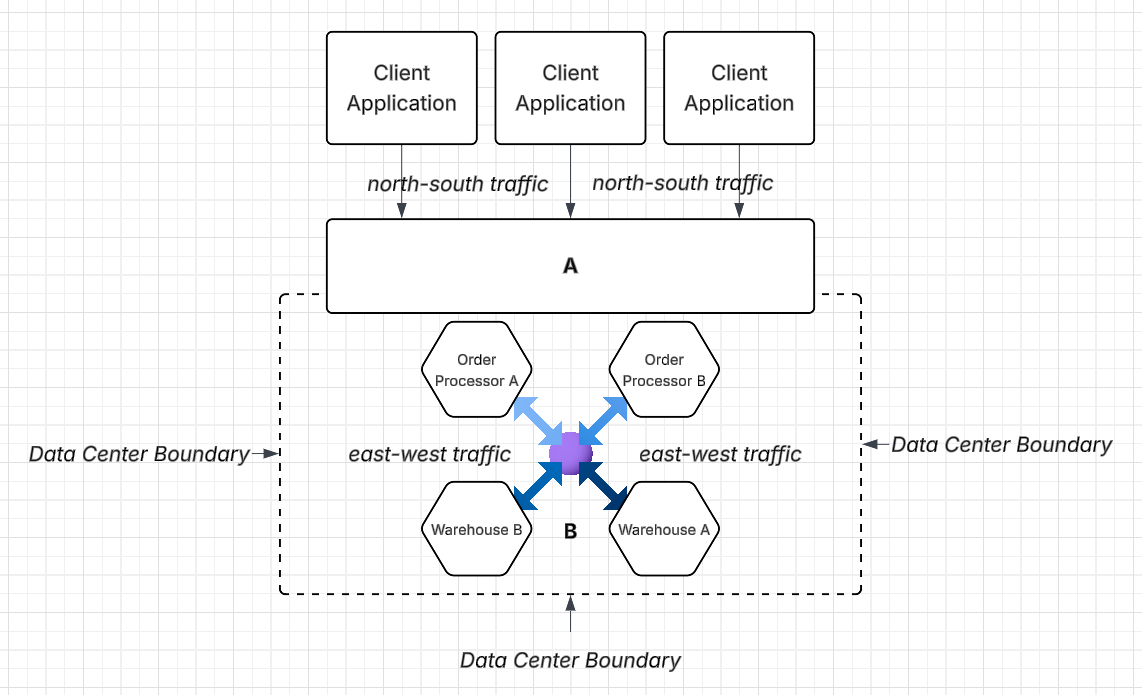
*The following diagram is a representation of the technology described in question 5….*



1. DNS records are often cached due to their Time to Live (TTL), which can result in outdated IP addresses being used. What technology can help in avoiding this issue by providing a consistent IP address (or endpoint) to reference instead of the individual IP addresses of microservice instances (**A** in the diagram above)?

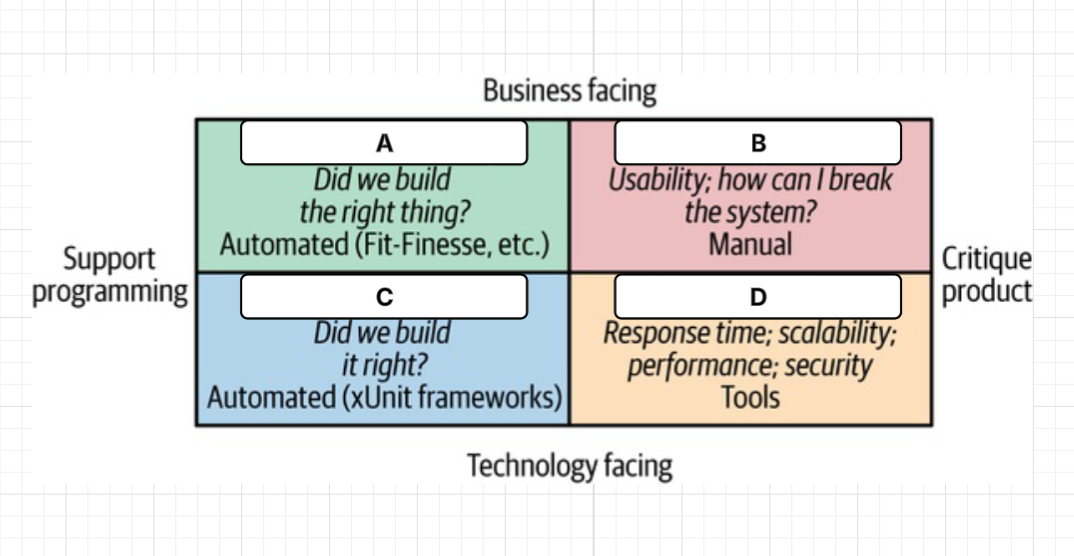
Load Balancer.

*The following diagram is a representation of the technologies described in questions 6 & 7….*



1. Focused more on north-south traffic, its main concern in a microservices environment is mapping requests from external parties to internal microservices (**A** in the diagram above).
2. Service Mesh.
3. API Gateway.
4. Common functionality associated with inter-microservice communication (east-west traffic) is pushed into it, reducing the functionality that a microservice needs to implement internally, while also providing consistency across how certain things are done (**B** in the diagram above).
5. Service Mesh.
6. API Gateway.

**Testing (22 Points).**

*For questions 8–11 (labeled A–D in the diagram below), complete Marick’s Testing Quadrant by matching each letter with the appropriate type of testing. Choose from the following options: Unit Testing, Exploratory Testing, Acceptance Testing, or Property Testing….*

1. (**A**).

Acceptance Testing.

1. (**B**).

Exploratory Testing.

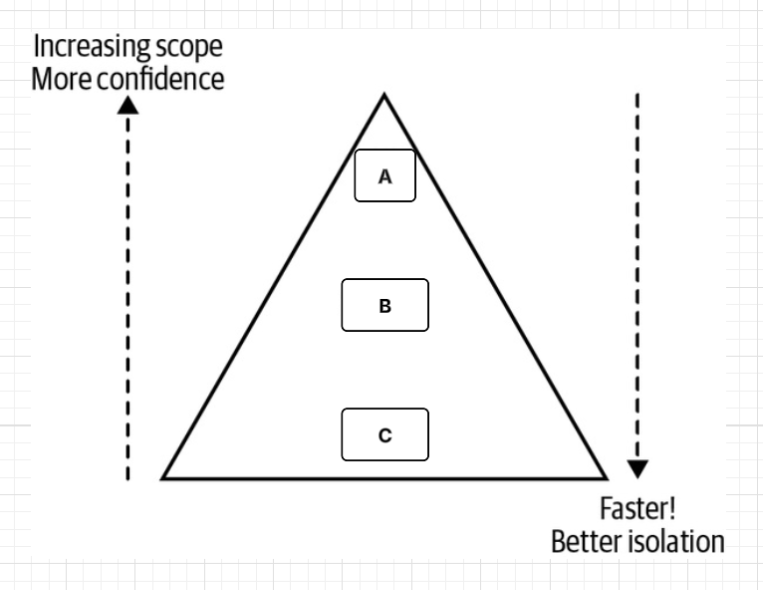
1. (**C**).

Unit Testing.

1. (**D**).

Property Testing.

*For questions 12–14 (labeled A–C in the diagram below), complete the Testing Pyramid by matching each letter to the correct testing scope. Choose from the following options: Unit Tests, Service and Contract Tests, or UI and End-to-End Tests….*



1. (**A**).

UI & End-to-End tests.

1. (**B**).

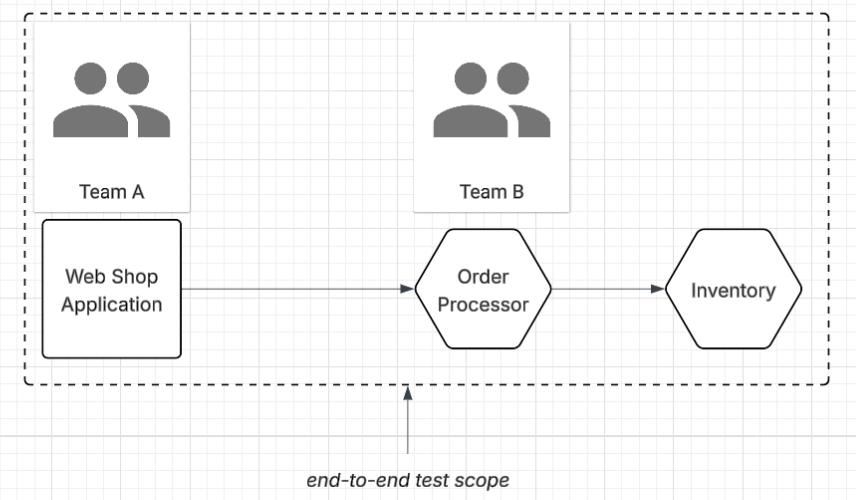
Service & Contract tests.

1. (**C**).

Unit tests.

1. A test creates a test version of a dependency, passes it into the system under test, and verifies that a specific method on the dependency was called exactly once with certain arguments. This test is using the dependency as a (choose one)?
2. Mock.
3. Stub.
4. A test creates a test version of a dependency, sets it up to return a specific value when called, and then checks whether the system under test returns the expected result. This test is using the dependency as a (choose one)?
5. Mock.
6. Stub.

*For questions 17 & 18 refer to the end-to-end testing scenario at MusicCorp (below)….*



At MusicCorp, **Team A** owns the **Web Shop** application, which allows customers to browse products and place orders. **Team B** owns the **Order Processor** and **Inventory** microservices. When a customer places an order through the Web Shop, the Order Processor validates and processes the order, and the Inventory service updates the stock levels accordingly.

The teams need to create an **end-to-end test** that validates the following flow:

* A user selects an item and places an order in the Web Shop.
* The Web Shop forwards the order to the Order Processor.
* The Order Processor confirms the order and triggers a stock update in the Inventory service.
* The Inventory service reduces the stock for the ordered item.
* The test verifies that the order is marked as “confirmed” and the inventory is updated correctly.

For each of the following ownership models — Shared Ownership and Single-Team Ownership — describe how the model would work in practice and identify potential downsides.

Consider factors such as:

* What happens when a test fails?
* How is responsibility handled?
* The overall number and quality of tests written.
* Maintenance and accountability over time.

1. Shared Ownership (Both Team A and Team B share ownership of the end-to-end test).

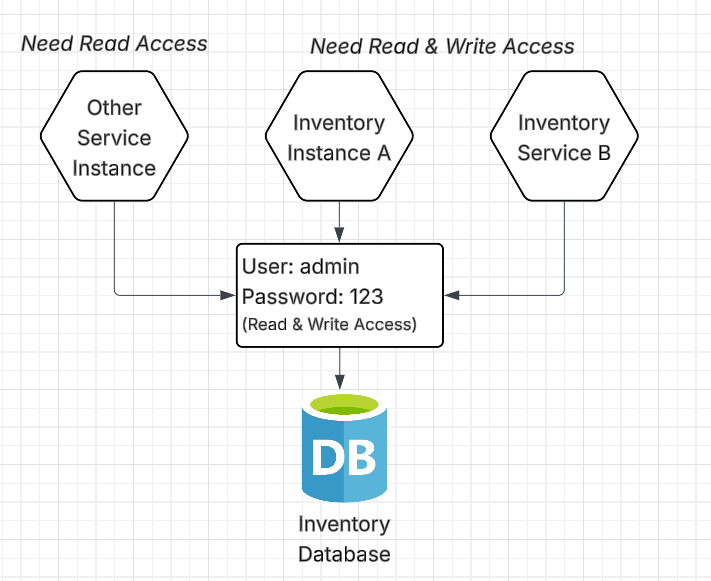
Tragedy of the commons.

1. Single-Team Ownership (Either Team A or Team B owns the end-to-end test entirely).

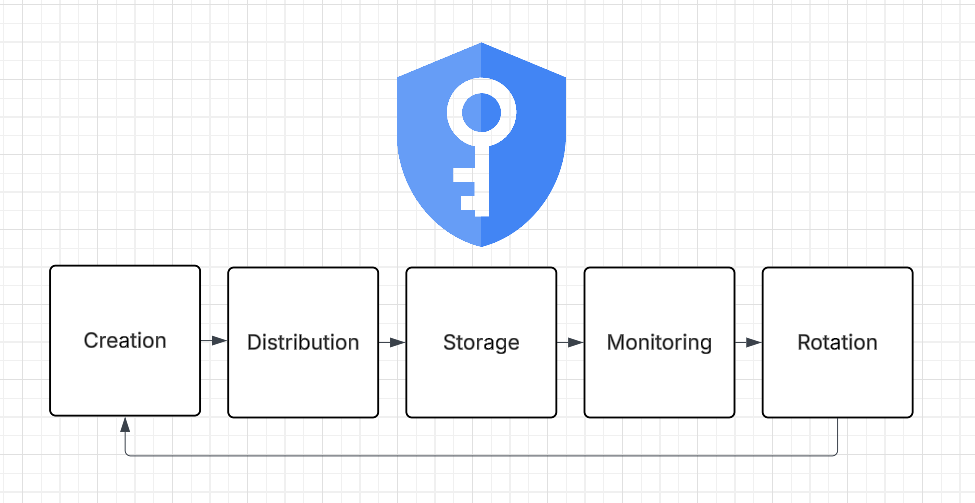
Test explosion or duplication of tests.

**Security (32 Points).**

1. Which of the following best distinguishes an application composed of a monolith as opposed to microservices in terms of security (choose all that apply)?
2. Larger attack surface.
3. Smaller attack surface.
4. Greater defense in depth.
5. Lesser defense in depth.
6. Which of the following best distinguishes an application composed of microservices as opposed to a monolith in terms of security (choose all that apply)?
   1. Larger attack surface.
   2. Smaller attack surface.
   3. Greater defense in depth.
   4. Lesser defense in depth.
7. Which of the following is NOT one of the five core functions of cybersecurity (choose one)?
8. Identify.
9. Recover.
10. Rebuild.
11. Respond.
12. True or false, the following application architecture follows the *Principle of Least Privilege*?



*For questions 23 & 24 identify the step in the “lifecycle of a secret” (below) described….*



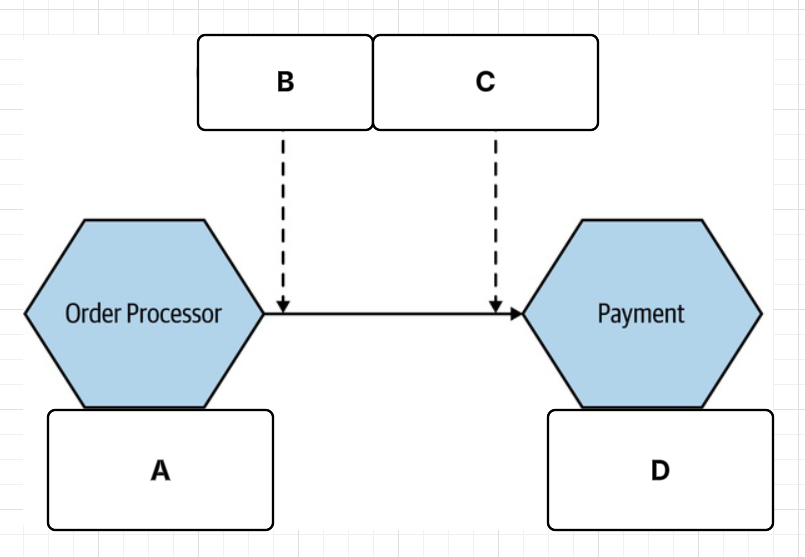
1. Are we able to change the secret without causing problems?

Rotation.

1. Is the secret kept in a way that ensures only authorized parties can access it?

Storage.

*For questions 25–28 (labeled A–D in the diagram below), complete the “four key concerns of data in transit” by matching each letter to the correct concern. Choose from the following options: Server Identity, Visibility of Data, Manipulation of Data, or Client Identity….*



1. (**A**).

Client Identity.

1. (**B**).

Visibility of Data.

1. (**C**).

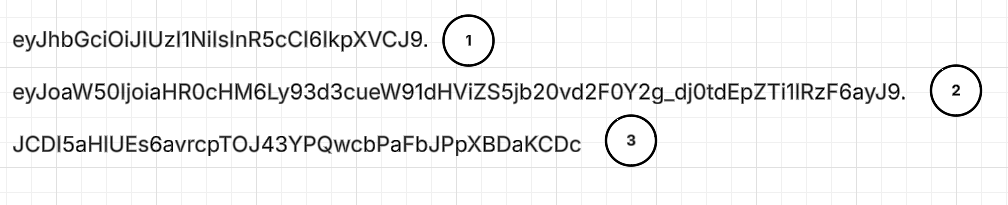
Manipulation of Data.

1. (**D**).

Server Identity.

1. Generally, when we’re talking abstractly about who or what is being authenticated, we refer to that party as which of the following (choose one)?
2. Principal.
3. Factor.
4. A piece of knowledge a user needs to authenticate themselves is referred to as which of the following (choose one)?
5. Principal.
6. Factor.

*For questions 31 – 33 match the portion of the token (****1****,* ***2****, or* ***3****) with its name…*



1. The payload.

2

1. The header.

1

1. The signature.

3

1. In the context of generating a signature (e.g., JWTs or HMAC), what is the value called that is combined with the header and payload to create the signature and verify data integrity and authenticity (choose one)?
2. Credential.
3. Secret.

**Resiliency (8 Points).**

1. An uptime of 99.999% (five 9’s) is an example of which of the following cross-functional requirements?
2. Response time/latency.
3. Availability.
4. Durability of data.

*For questions 36–38, choose the appropriate type of CAP* (Consistency, Availability, and Partition Tolerance) *system (****CA, CP, or AP****) that best fits each scenario described….*

🧠 **Scenario 1:**

MedicalCo runs a system for handling electronic prescriptions between doctors and pharmacies. When a doctor submits a prescription, it's critical that the exact same information is visible to the pharmacy—no outdated or conflicting data is acceptable. If there’s a temporary network issue between systems, the prescription data may be delayed rather than risking inconsistencies.

1. Given this behavior, which type of CAP system does **Scenario 1** represent (justify your answer)?

CP

🧠 **Scenario 2:**

MedicalCo offers a mobile app and website that allows patients to book and manage appointments. The system is distributed across multiple data centers for reliability and speed. If one data center is isolated due to a network issue, patients in that region can still book appointments, even though the bookings might temporarily conflict or need syncing later.

1. Given this behavior, which type of CAP system does **Scenario 2** represent (justify your answer)?

AP

🧠 **Scenario 3:**

At each MedicalCo clinic, patients check in via on-site kiosks connected to a local server. These kiosks always display consistent information and are available for use if and only if the clinic's internal network is operational. However, if the local server fails or the clinic loses power, the kiosks become unusable.

1. Given this behavior, which type of CAP system does **Scenario 3** represent (justify your answer)?

AC – Single process monolith.