Digital Dim Sum: Mastering Microservices for Many Palates

DeveloperWeek 2023
API & Microservices Track
February 17, 2023

Spencer Carver
Software Engineer, Bloomberg Media

TechAtBloomberg.com

Hello everyone and welcome to “Digital Dim Sum: Mastering Microservices for Many Palates”. I hope this mouthful of a title was enough to whet your appetite and pique your curiosity for the conversation to come, but before we dig in, a brief introduction.

About Me

spencerrc

@spencerrc@masto.nyc

https://spencer.carvers.info

About Me

Software Engineer in Bloomberg Media since 2015, supporting www.bloomberg.com

Focused on digital subscriptions since 2018

TypeScript / React enthusiast

Enjoys:

- Visual documentation & diagrams
- Food puns
- Forcing tenuous analogies

Bloomberg

Engineering

My name is Spencer Carver, and I'm a software engineer in Bloomberg Media. Our flagship product is bloomberg.com, and for the past 5 years I've been responsible for supporting the growth efforts of our digital subscriptions business on the web. In this time I've been fortunate enough to be able to experiment with a wide range of technologies and organizational approaches all aimed at addressing the ever-changing data needs of our news business. And so today I thought it would be fun to share some of these learnings with all of you through a stretched metaphor about food.

—

Reference: spencer.carvers.info

Reference: www.bloomberg.com

Reference: Tech at Bloomberg

Objective

Objectives of this talk

- Introduce the “Four Levels” framework from the Epicurious YouTube series
- Address each level via a food analogy that relates to Software Design Patterns around Application Architecture
- Discuss each level and applied learnings in order to answer the following questions:
    - Where are you now?
    - Which level is right for you?

More specifically, I’d like to borrow the “Four Levels” framework from the popular Epicurious YouTube channel, and apply it to discussing software design principles and application architecture. We’ll use this structure to evaluate a hypothetical product at each “Level”, and contextualize aspects of the approach that work, as well as those that don’t. Finally, after discussing each case, I’ll try and draw some conclusions that you can take away in the event that anything in this talk sounds familiar.

—

Reference: Four Levels: Pancakes

Level 4: Food Scientist

- Provides Analysis and Meta-Commentary between each approach below

Level 1: Amateur

- Sufficient Solutions
- Limited Exposure to Methods

Level 2: Home Chef

- Familial Knowledge
- Expertise in a Limited Area

Level 3: Professional

- Broad Understanding
- Breadth of Experience

4 levels logo referencing the Epicurious series created from pierogi line art by Lucas Collins on pngitem.com.

The premise of the “Four Levels” series is to challenge 3 chefs of different skill levels to make the same dish. The levels range from amateur to professional, and then a food scientist reviews the process each has taken and explains why techniques applied by each chef worked well, or where they could have better leveraged an approach of one of their peers.

So, keeping this framework in mind, I’d like to ask you to…

—

Reference: Four Levels: Pancakes

Consider: Dumplings

- Well defined components
- Can be a stand-alone meal or part of a larger dim sum experience
- Evaluated on many different aspects
  - Flavor
  - Cost
  - Preparation Time
  - Appearance

… consider dumplings. While I find it fun to philosophically consider dumplings, they are also a great choice of dish for the purpose of discussion within the “Four Levels” framework!

They are simple to start, difficult to master, and while there are many variations, there are also consistent expectations about what constitutes a quality dumpling.

For each level in our framework, we’ll consider the dumpling from that chef’s perspective, and see how that outlook translates into our hypothetical software product...

Dumpling Academy

Filling you up on the freshest recipes

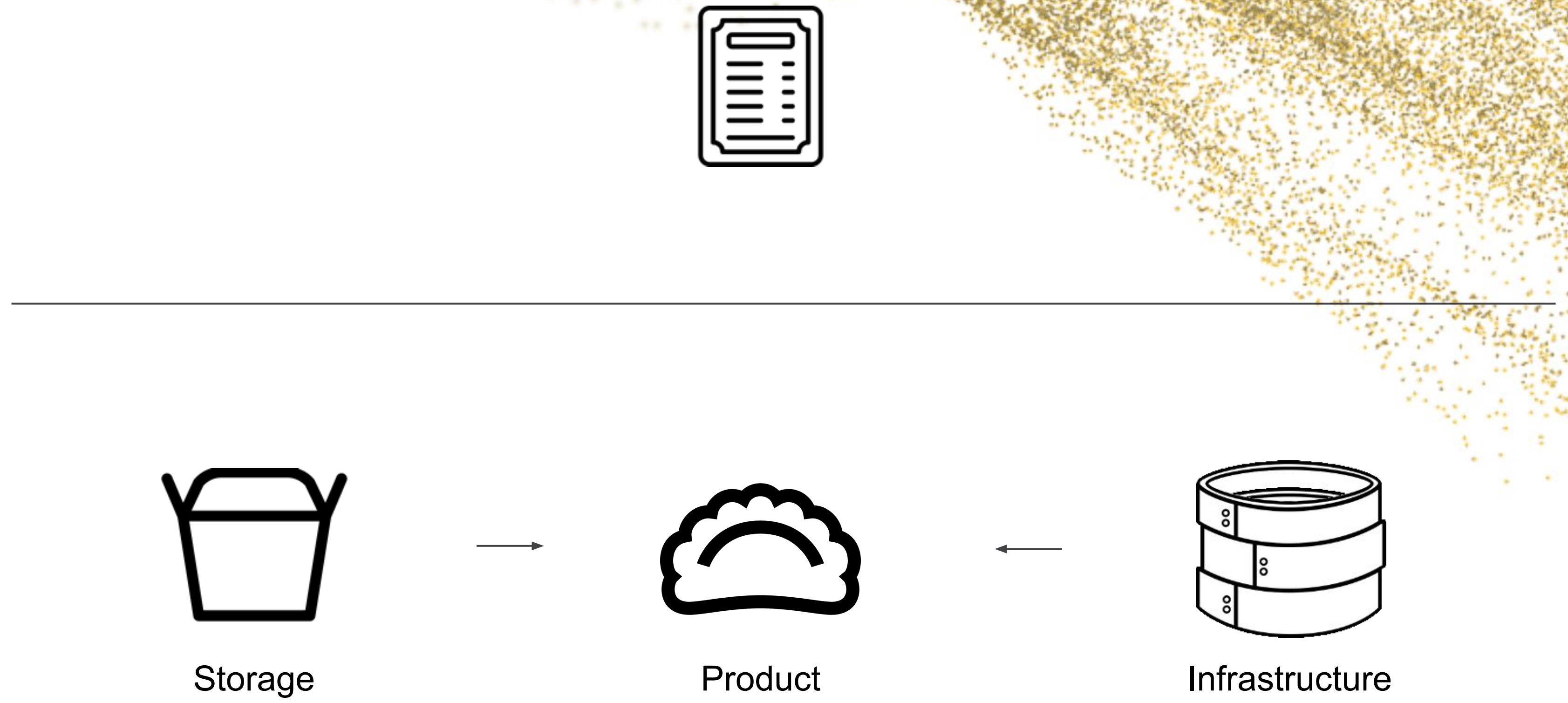
…Dumpling Academy, a recipe website whose goal is to become the definitive dumpling resource.

And so, with our prep work out of the way for both food and technology discussions, let's begin cooking…

Level 1: Amateur

…alongside our Level 1 chef. While they may bear the label “Amateur”, that doesn’t mean they don’t know what to do. First and foremost, the level 1 chef is concerned with producing their desired dish, and everything else is secondary. A singular focus coupled with lack of experience can lead to many pitfalls, such as overworking the dough, improperly seasoning the filling, or misjudging the cook time, but the mindset lends itself to many positive cases as well.

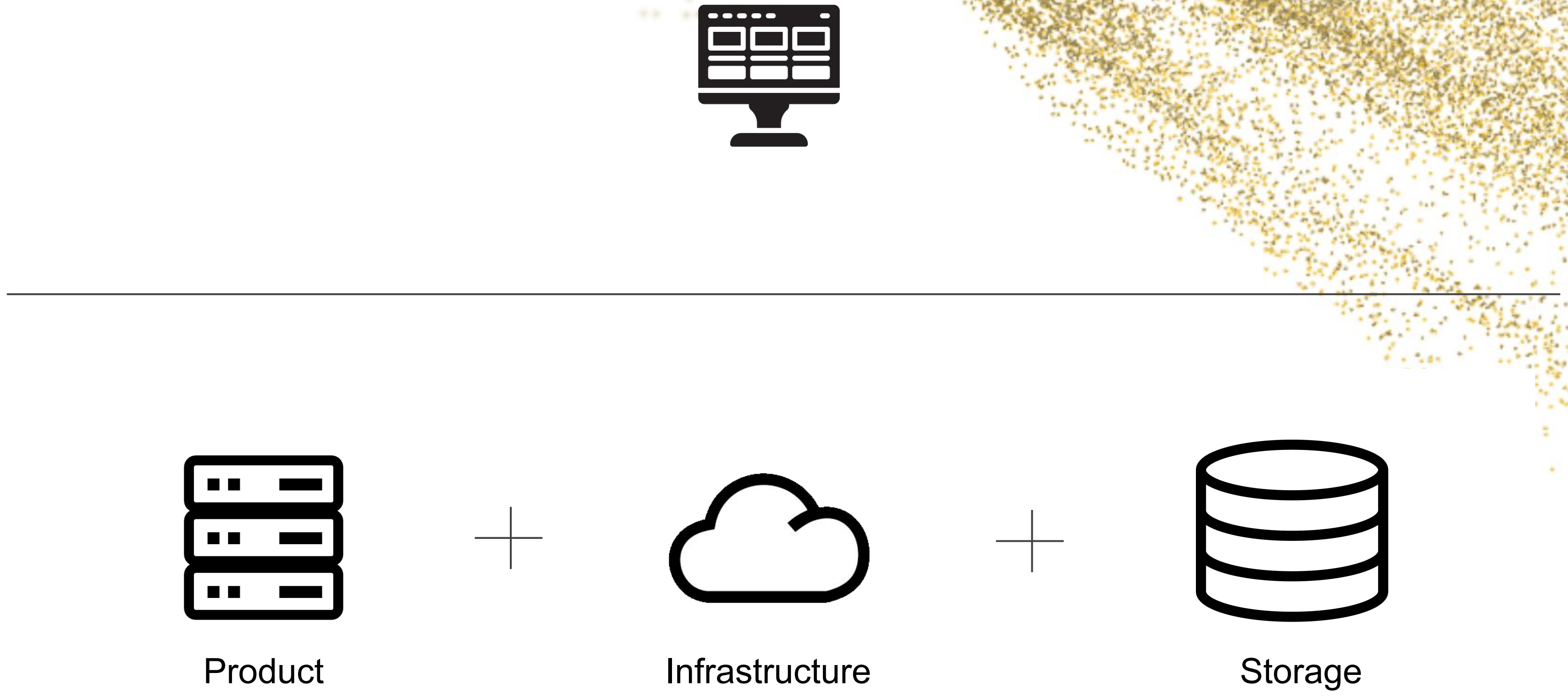
Level 1



steamer adapted from art by stephanie wauters for The Noun Project

A holistic approach means that, given the freedom to choose any equipment they need, the level 1 chef will choose the best option for their dumplings. Likewise for serviceware and storage, each choice the chef makes is made in service of the product's current needs. If those needs don't change with time, the chef has already obtained the right tools and experience producing the product they want, and can focus on further optimizations rather than needing to revise their recipe or process.

Level 1: Monolith



database icon by zwicon on svgrepo.com

There’s a holistic software design pattern as well, one you may already be familiar with in some capacity; the monolith. When done correctly, an application leveraging a monolithic architecture will ensure that each choice being made benefits the end product.

Level 1: Monolith

Dumpling Academy

Product

Recipe

```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
…
```

Person

```
id: guid;
name: string;
recipes: Recipe[];
bio: string;
…
```

Dumpling Academy as a monolith could make a lot of sense. We're concerned only about dumpling recipes and getting them into the hands of chefs worldwide as fast as possible. Recipes are well structured and don't change much after they are published, and while we may have other types of content that we need to manage, they're complimentary in towards our recipe data.

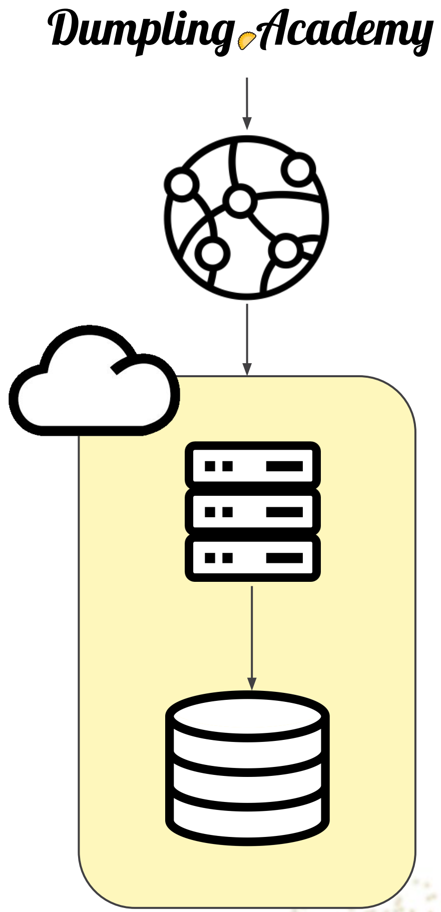
Level 1: Monolith

Pros

- **Developer Experience**
  - Debugging
  - Deployment
  - Development
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Performance**
  - Speed
- **Networking**
  - Latency

Cons

- **Management**
  - Organizational Overhead
  - Ownership
- **Performance**
  - Reliability
- (lack of) **Logical Isolation**
- **Infrastructure**
  - Scalability
  - Technical Overhead
- **Growth?**



database icon by zwicon on svgrepo.com

Leveraging a monolith benefits Dumpling Academy in other ways as well. With everything being co-located, developer experience is about as streamlined as it can be. A single well-maintained codebase allows debugging and testing tools to work without the need for much customization, providing a lower barrier to getting new developers up and running. It is also ideal for enforcing consistent coding standards in a project, since you are guaranteed to be using a configuration from a single location. A well-structured monolith can minimize external networking calls, ensuring a speedy response.

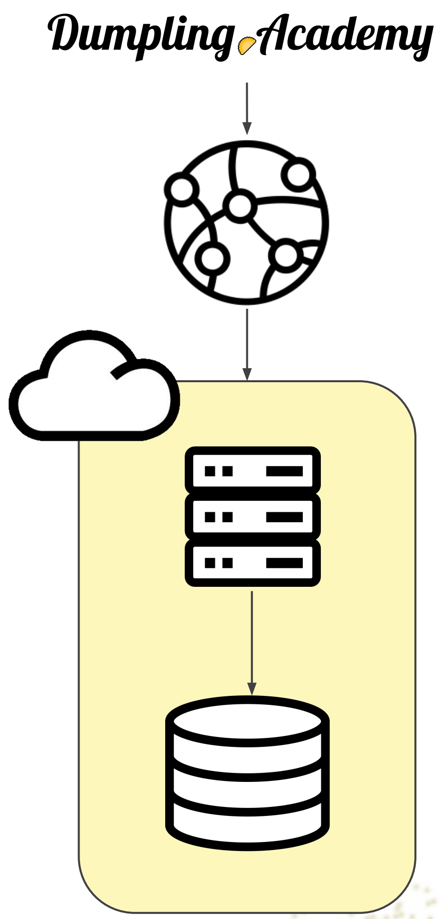
Level 1: Monolith

Pros

- **Developer Experience**
  - Debugging
  - Deployment
  - Development
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Performance**
  - Speed
- **Networking**
  - Latency

Cons

- **Management**
  - Organizational Overhead
  - Ownership
- **Performance**
  - Reliability
- (lack of) **Logical Isolation**
- **Infrastructure**
  - Scalability
  - Technical Overhead
- **Growth**?



But while it "can" scale well, doesn't mean it does so in all cases. A singular codebase can create congestion if multiple features are being developed simultaneously, and if multiple teams are supporting a single monolith, ownership becomes amorphous. Any defects introduced can crash our entire recipe site, even if we were only trying to make additions to our secondary flows, such as the authoring chef's biography page.

Lastly, we may have missed our mark on the product itself. Dumpling Academy may have started focused only on recipes related to dumplings, but aside from search how will visitors find content? We may have started by promoting the newest recipes, or even the most popular, but perhaps we're better served allowing a recipe editor to manually curate our landing pages.

Pain Points?

If one or more of the following are true, it may be time to reconsider the monolith as your preferred approach:

1. The monolith is responsible for many unrelated concerns

Dumpling Academy HOT LIST

2. The technology of the monolith is outdated and/or causing problems

3. There are changes that can be introduced (e.g. dynamic data) that can cause problems in unrelated areas

Dumpling Academy Guest Recipes

4. Certain code paths within the monolith are disproportionately used

So what are some signs that a monolith may not be the right choice for our site? You may have picked up on the use of qualifiers earlier when explaining why this approach benefited Dumpling Academy. "Well-maintained". "Well-structured". There is a cost to keeping those statements true, especially if our site is growing and changing.

If uniformity of experience is no longer a core tenant of the product, that is also a good indicator that a monolith may no longer be the right pattern. The moment Dumpling Academy introduced a configurable landing page, even though it was showing recipe content, we started down a slippery slope.

As time goes on the maintainers of Dumpling Academy change, and so too does our common technology stack. Even if our product is still manageable for now, if the requisite knowledge to continue operation isn't something that is easy to find, being locked in to the monolith will create more opportunities for problems.

Level 1: Monolith

Pain Points?

If one or more of the following are true, it may be time to reconsider the monolith as your preferred approach:

1. The monolith is responsible for many unrelated concerns
2. The technology of the monolith is outdated and/or causing problems
3. There are changes that can be introduced (e.g. dynamic data) that can cause problems in unrelated areas
4. Certain code paths within the monolith are disproportionately used

Dumpling Academy HOT LIST

Dumpling Academy Guest Recipes

We may be given opportunities to leverage syndicated content from other recipe websites, and while this can be a great opportunity to provide benefit for our users, that content is likely not going to conform to the format of our own recipes. Any problems loading this external content could be at best a slightly degraded or broken experience, and at worst bring down our entire site.

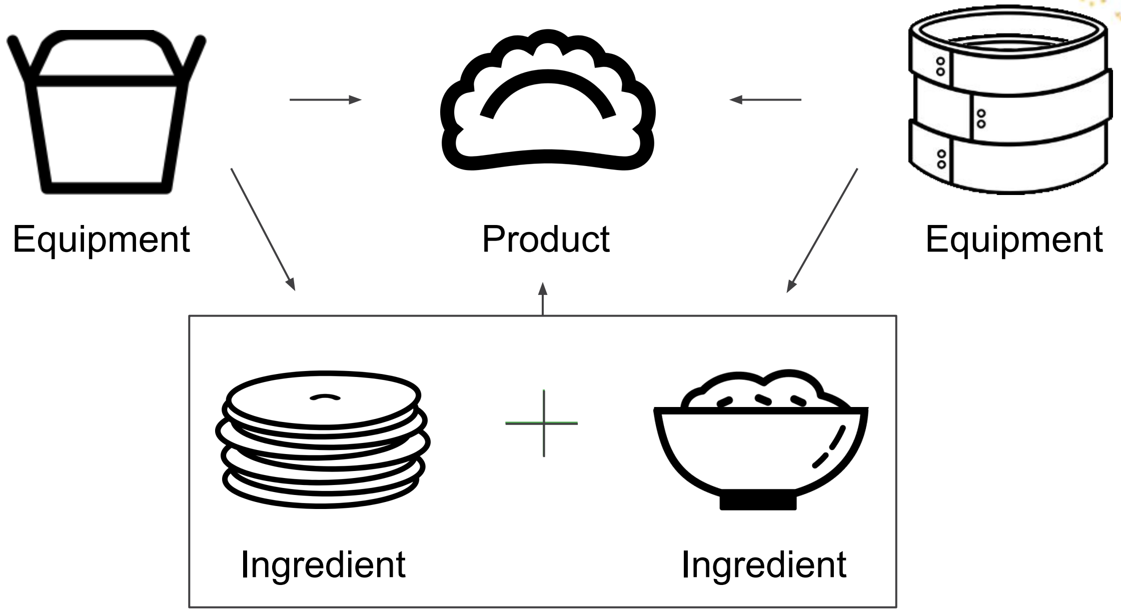
And lastly, certain parts of our site are just used more. If we were to start providing real-time ingredient sourcing information on our recipe pages, it's very easy to see it as an extension of the monolith, when in reality its functionally a different product and should be treated as such.

But if the needs of our site don't fit with this approach, what does that mean we're growing to? Let's check in with our level 2 chef.

Level 2: Home Chef

The “Home Chef” has realized that a single holistic view just doesn’t cut it for the dumplings they want to create. Instead, they view the problem from a compartmentalized perspective, treating each component of their dumpling as the final product of a separate process. This allows for greater flexibility at the cost of organizational overhead, but the level 2 chef purposefully chooses a structure that benefits them.

Level 2: Mise En Place



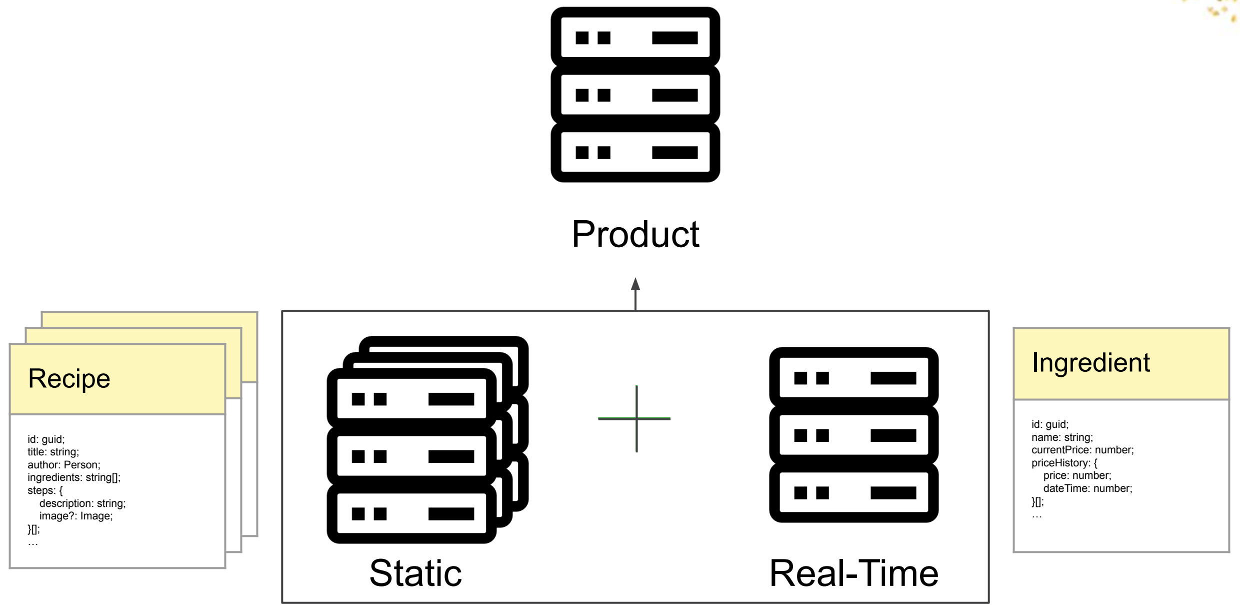
steamer adapted from art by stephanie wauters for The Noun Project
wrappers adapted from tortilla image from flaticon.com

Focus on the ingredients as intermediate products means more interconnected concerns, but while the equipment is still important, it’s not a different concern than our level 1 chef dealt with, just more common, so we’ll set gloss over it for now. In the culinary world, the process by which ingredients are prepared and laid out before cooking is referred to by the term “mise en place”, and there is an applicable software design pattern that follows the same practice.

If you hadn’t guessed, that approach is …

Level 2: Microservices

Dumpling Academy



```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
    description: string;
    image?: Image;
}[];
…
```

```
id: guid;
name: string;
currentPrice: number;
priceHistory: {
    price: number;
    dateTime: number;
}[];
…
```

… microservices! So what will Dumpling Academy look like within a microservice-oriented architecture? We ran into problems when non-recipes such as person and curation data started being treated as primary concerns on their own pages, so we want to split out each into their own service. Likewise, the notion of entertaining real-time data is a fundamentally different concern, and so to should have it’s own service as well.

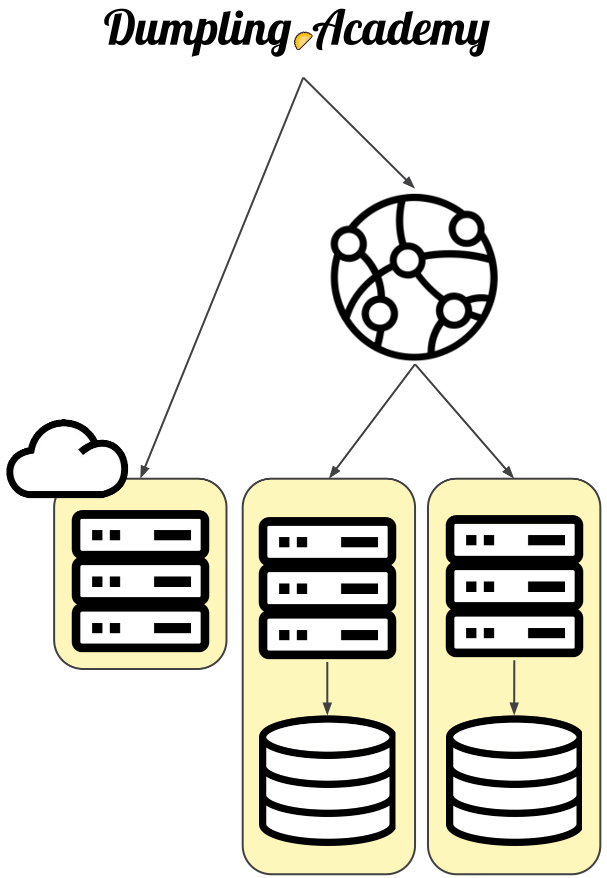
Level 2: Microservices

Pros

- **Developer Experience**
  - Deployment
  - Development
- **Management**
  - Ownership
- **Performance**
  - Reliability
- **Logical Isolation**
- **Networking**
  - Load
- **Growth**

Cons

- **Developer Experience**
  - Debugging
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Management**
  - Organizational Overhead
- **Infrastructure**
  - Cost
  - Technical Overhead
- **Networking**
  - Latency



database icon by zwicon on svgrepo.com

This compartmentalization brings back some of the perks we saw with a well-maintained monolith, as each microservice can easily be developed and deployed independently; making it easier to revise large swaths of our curation-focused pages without even touching the recipe service at all!

Ownership with this approach can also be much clearer, as teams can be responsible for all aspects of a single service, while still maintaining their ability to execute independently. This leads to greater confidence in the service correctly handling all use cases before releasing new features or adding another consumer. This structure also allows us to create multiple instances of the recipe service, which is used on more content pages than the others, with a smaller impact than if we needed to scale everything.

And perhaps most importantly, the microservices pattern sets us up well for continued growth. A new domain can be created as an independent service without concern for impact to existing pages.

Level 2: Microservices

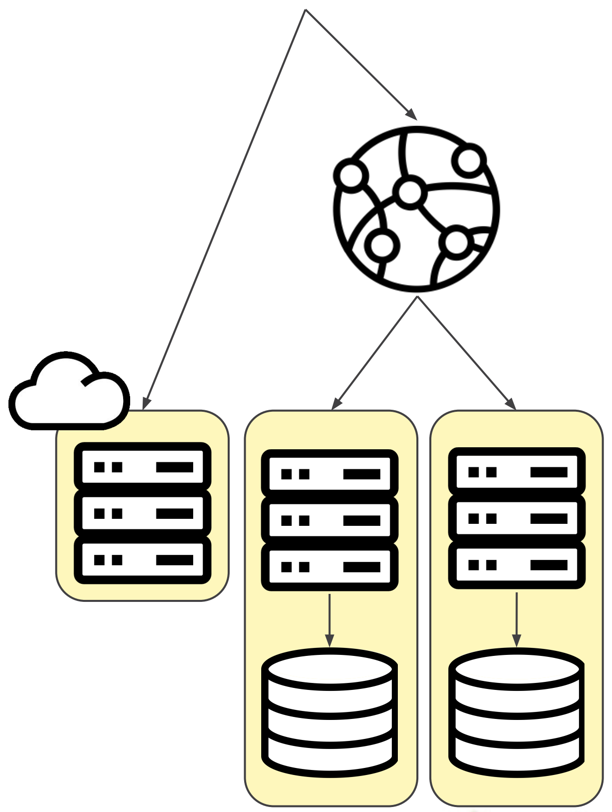
Pros

- Developer Experience
  - Deployment
  - Development
- Management
  - Ownership
- Performance
  - Reliability
- Logical Isolation
- Networking
  - Load
- Growth

Cons

- **Developer Experience**
  - Debugging
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Management**
  - Organizational Overhead
- **Infrastructure**
  - Cost
  - Technical Overhead
- **Networking**
  - Latency

Dumpling Academy



database icon by zwicon on svgrepo.com

But while each service operates well in isolation, our product is still comprised of all of these, and there are perils to this organizational complexity. While some aspects of developer experience are improved, others require more work, such as validating integrations and workflows that cross multiple systems. With microservices owned by separate teams, it becomes more challenging to maintain the same level of code standards, and teams may even choose to use different languages or technologies to best support their own work.

Management across teams requires coordination and understanding of the full product, which for a large recipe website is not a trivial ask. Running 4 microservices providing data in addition to our application means at least 5x the infrastructural investment, and separate systems introduce networking call time which will slow down the user-facing response.

Pain Points?

If one or more of the following are true, it may be time to consider an effort to revise your afflicted microservices:

1. A microservice has become monolithic
2. Associated infrastructure costs are too high
3. Dependencies between microservices causing problems
4. Time loss when updating consumers to service changes
5. New features for the product are infeasible or difficult due to logical sprawl

Microservices work well when they are organized properly around the product they support, and just as when discussing the monolithic approach earlier, there is a cost to ensuring that the services supporting your application are compliant. Pain points encountered with the microservices-oriented Dumpling Academy are largely due to the real-world needs of the product mismatching with the organizational expectations and not being addressed in time. What can that look like?

One of our microservices becoming a monolith in it’s own right is certainly one possibility, though this can be managed with the introduction of additional microservices as well.

However more microservices means more cost, and it may in fact be a worthwhile decision to intentionally take on some developmental pain points in order to be able to operate at all.

Pain Points?

If one or more of the following are true, it may be time to consider an effort to revise your afflicted microservices:

1. A microservice has become monolithic
2. Associated infrastructure costs are too high
3. Dependencies between microservices causing problems
4. Time loss when updating consumers to service changes
5. New features for the product are infeasible or difficult due to logical sprawl

Dumpling Academy PRO

As Dumpling Academy grows our user base larger, we may want to consider a premium feature, which would mean introducing authentication into many aspects of our site. While it may seem like a good idea to leverage a single microservice for authentication purposes, that pattern can very easily backfire, with multiple microservices re-checking credentials repeatedly in a single call path.

While microservices excel for managing our separate concerns, adding new attributes also requires each consuming application to update. Adding something as simple as a link to a twitter handle on the person object could require dozens of updates in order to make it somewhere user-facing. And with so many changes related to adding a single field, developmental complexity can skyrocket for more complicated features.

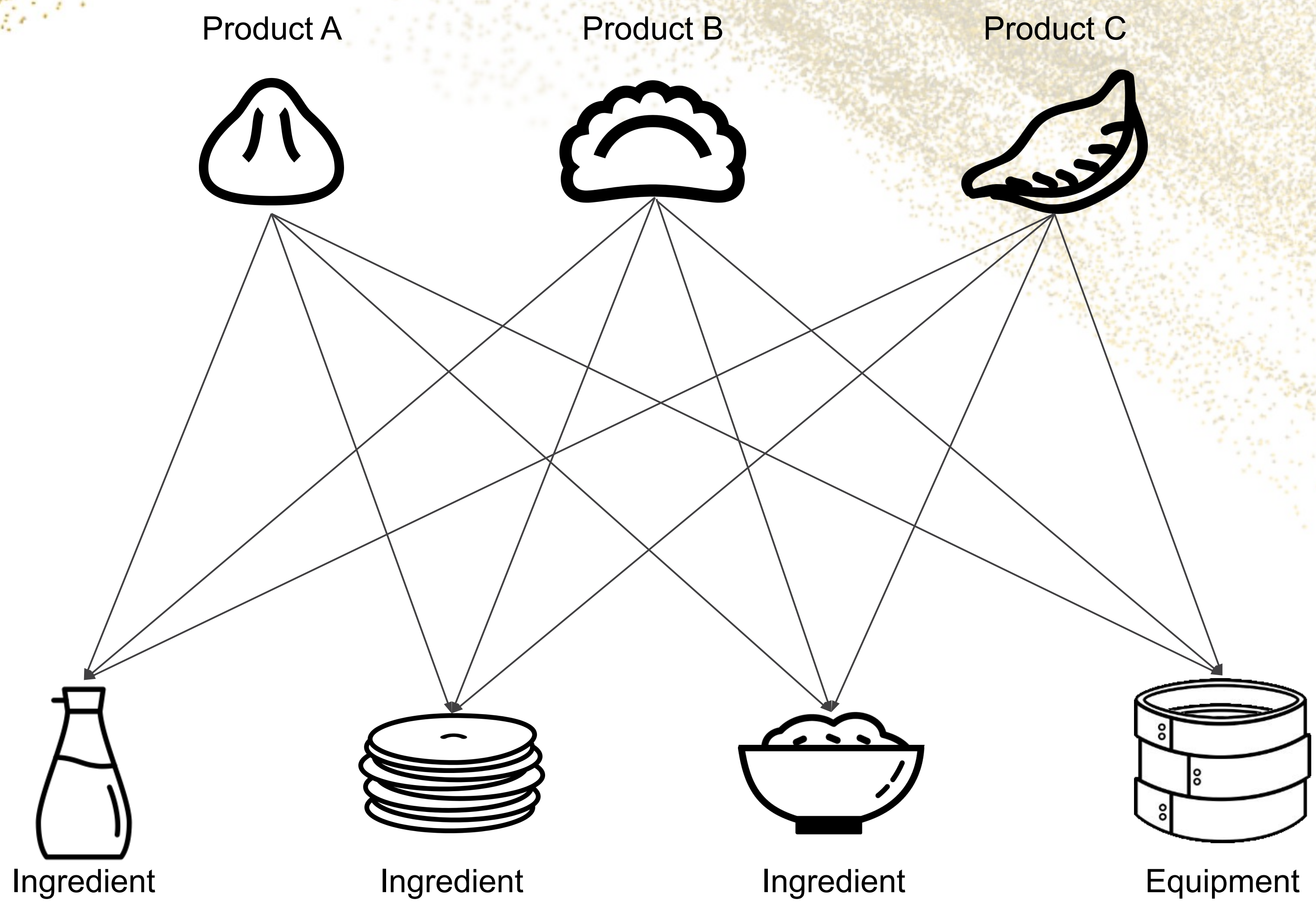
While compartmentalization has solved some problems, others continue to persist, so how would our level 3 chef approach this problem?

Level 3: Professional

Before we answer that question, lets understand the mindset of our "Professional". The level 3 chef may not be trained specifically to make dumplings, but they are trained to understand the aspects of what makes them good. Our chef understands why certain flavor combinations are traditional, and what spices can be added or withheld to for maximum effect. The level 3 chef is actually more likely to experiment and deviate from existing procedures than the level 2 chef, because they are looking to explore new things to see what alteration may improve their dumplings further.

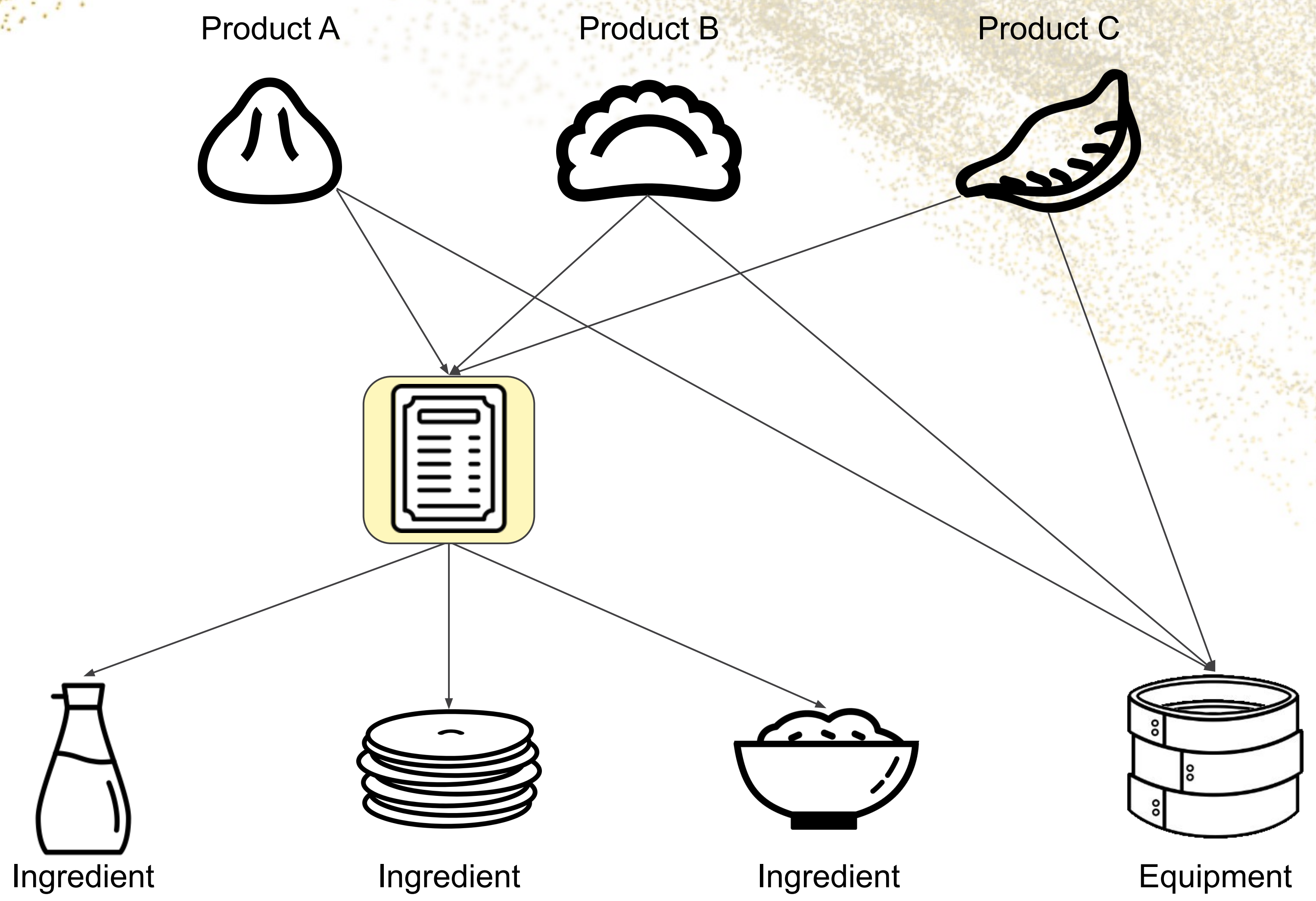
But despite some differences in focus the level 3 chef is still leveraging compartmentalization, just on a different scale. The chef's exploration of variety means that each ingredient isn't optimized just for one type of dumpling, but perhaps many. The level 3 mindset can be described as adaptive, able to flex between multiple approaches in order to choose the best option for any particular product at a given time.

Level 3



Each version of dumpling needs a slight variation of each ingredient, and the result is a large interconnected web of dependencies. In order to ensure everything operates smoothly in the kitchen while enabling the best version of each dumpling, the level 3 chef will set aside some additional space to use for the purpose of assembly.

Level 3

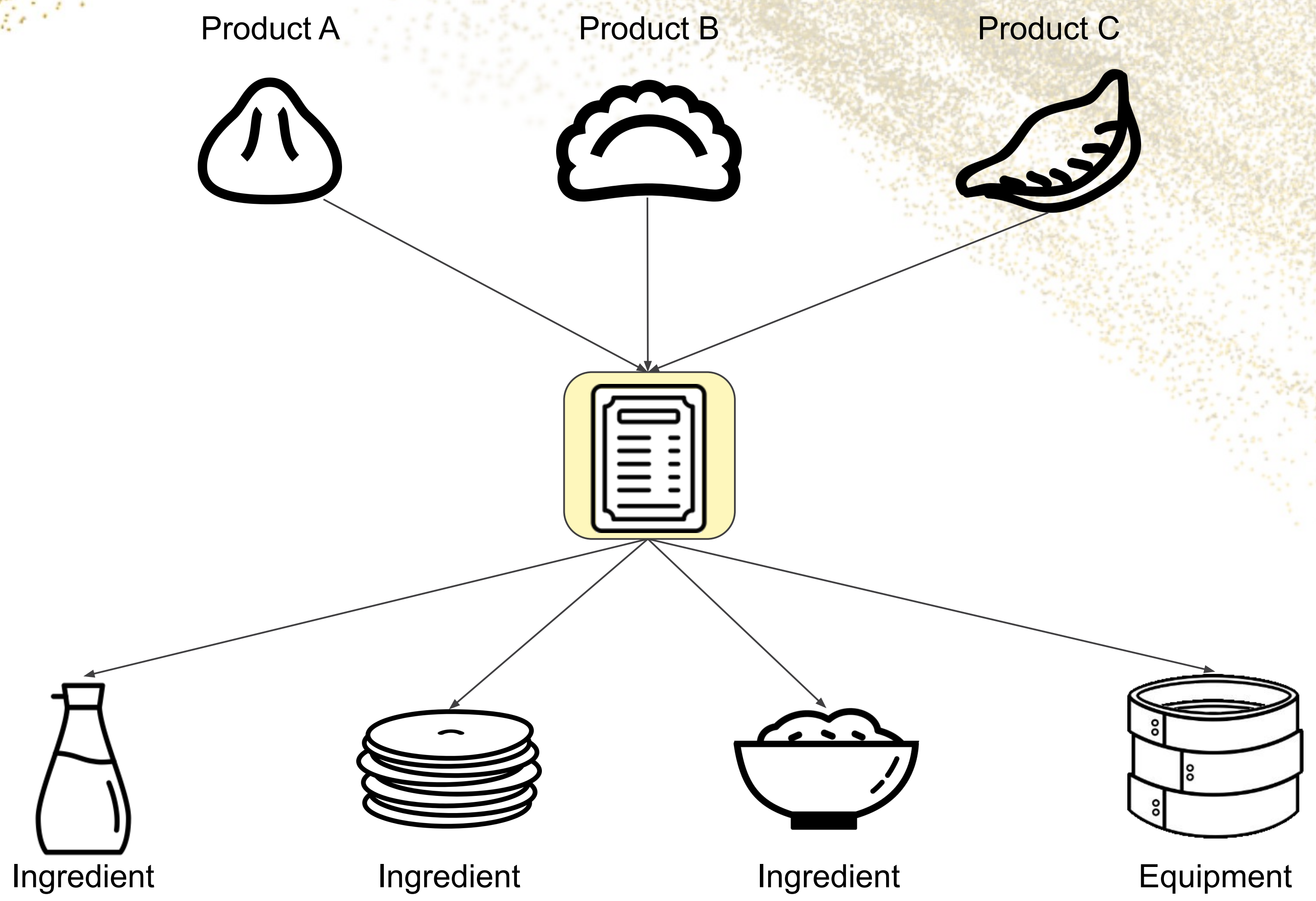


steamer adapted from art by stephanie wauters for The Noun Project
soy sauce, wrappers (adapted from tortilla) images from flaticon.com

This reserved area allows our process to change from each product being responsible for collecting all of its ingredients, to being able to request the specific variations they need. All ingredients can be returned from the prep area at one time before being used for successive steps in our preparation.

But there's no reason that the preparation area can only be used to collect discrete ingredients…

Level 3

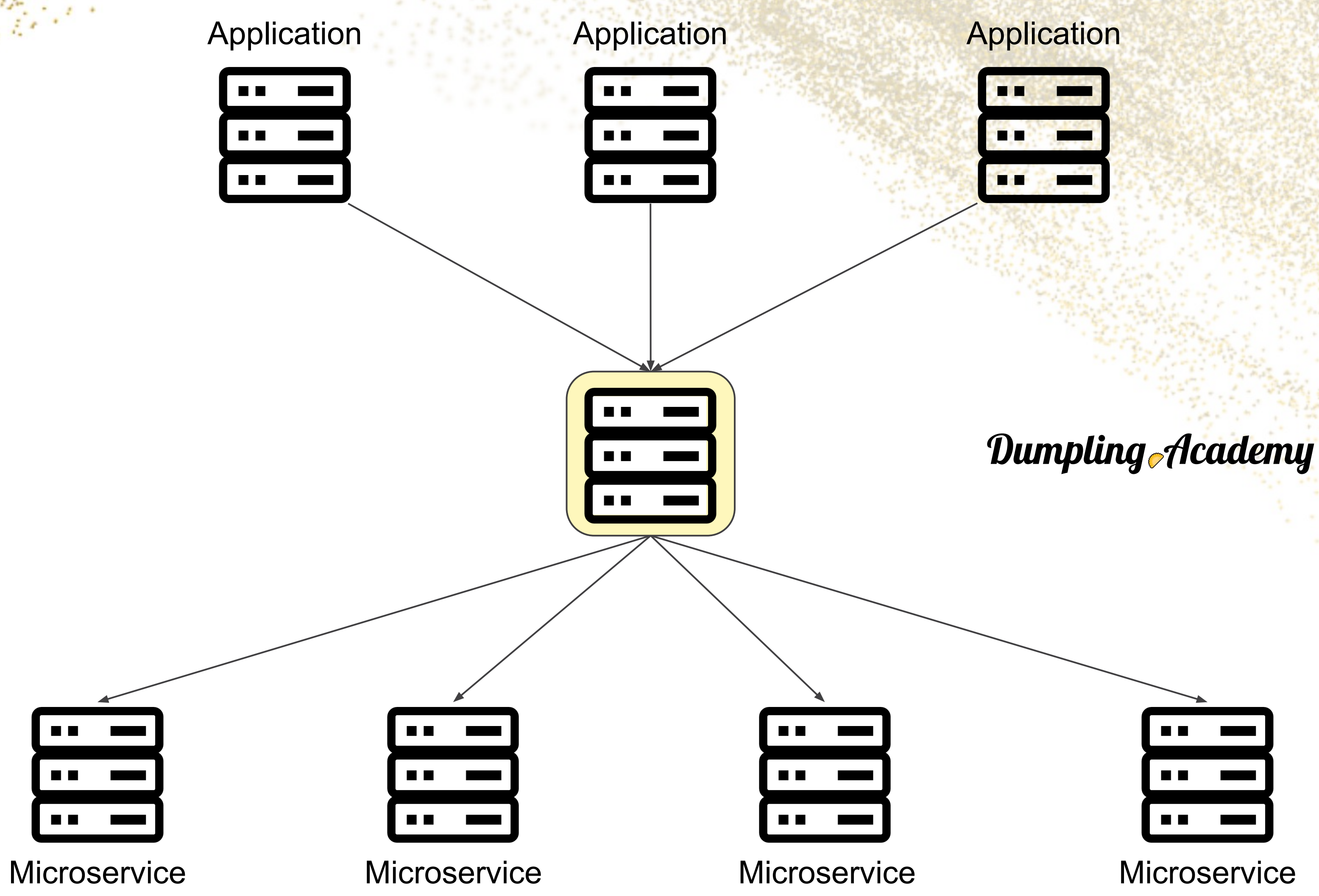


steamer adapted from art by stephanie wauters for The Noun Project
soy sauce, wrappers (adapted from tortilla) images from flaticon.com

…we can pass back our assembled, uncooked dumpling as well!

A well-managed preparation area provides a critical resource to a professional kitchen, further leaning into the organizational constraints that we already saw benefited our level 2 chef's approach earlier. The same is true for our software design patterns…

Level 3

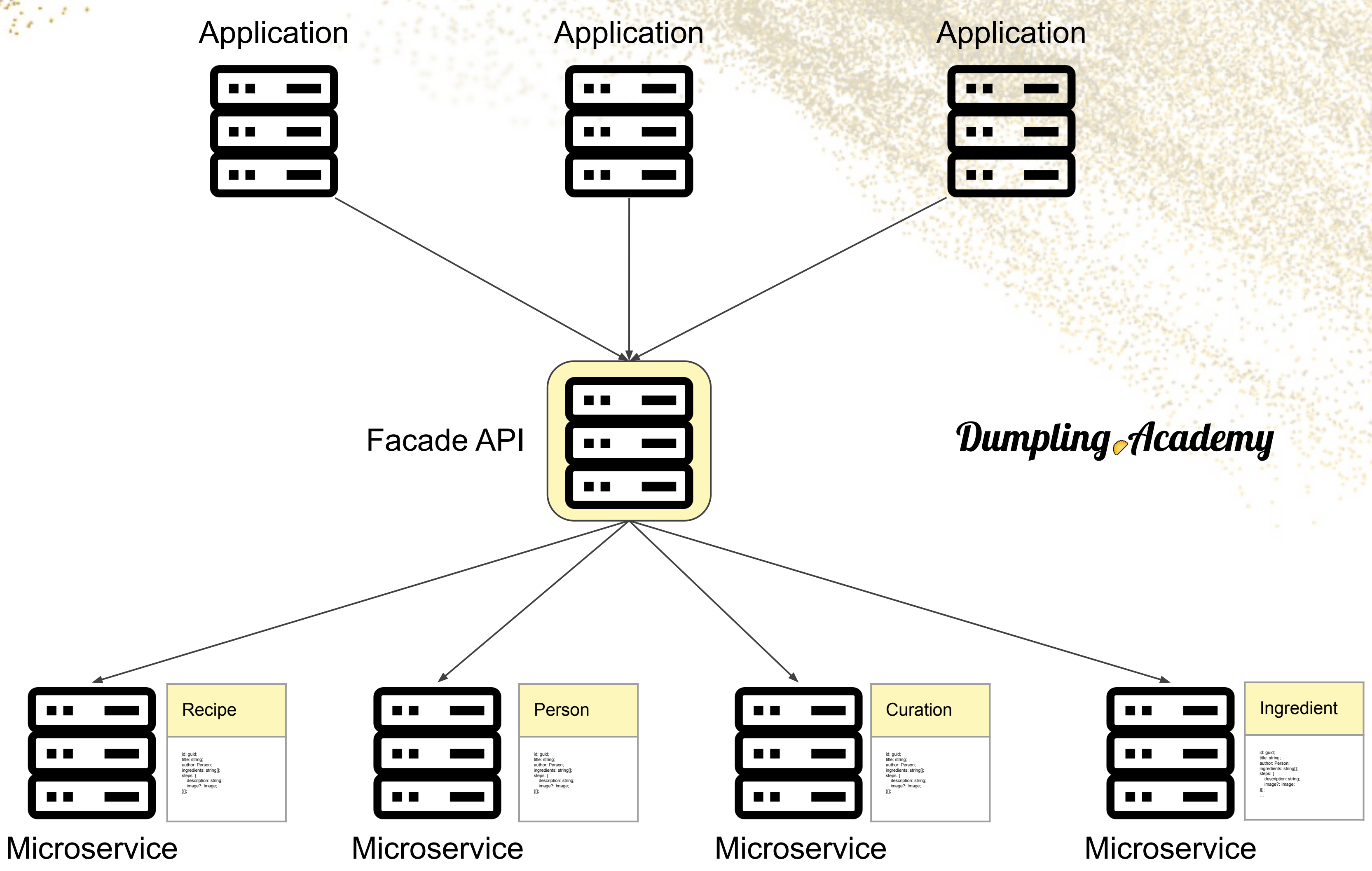


Dumpling Academy

…where we can extend our microservice pattern for Dumpling Academy, and add an intermediate application to help simplify the need for multiple applications to have knowledge of all dependent changes, instead only needing to understand how to talk to one system.

And at this point you may be thinking that there’s no reason that we couldn’t leverage this approach at level 2 as well, and you’d be correct!

Level 2: Microservice Facade

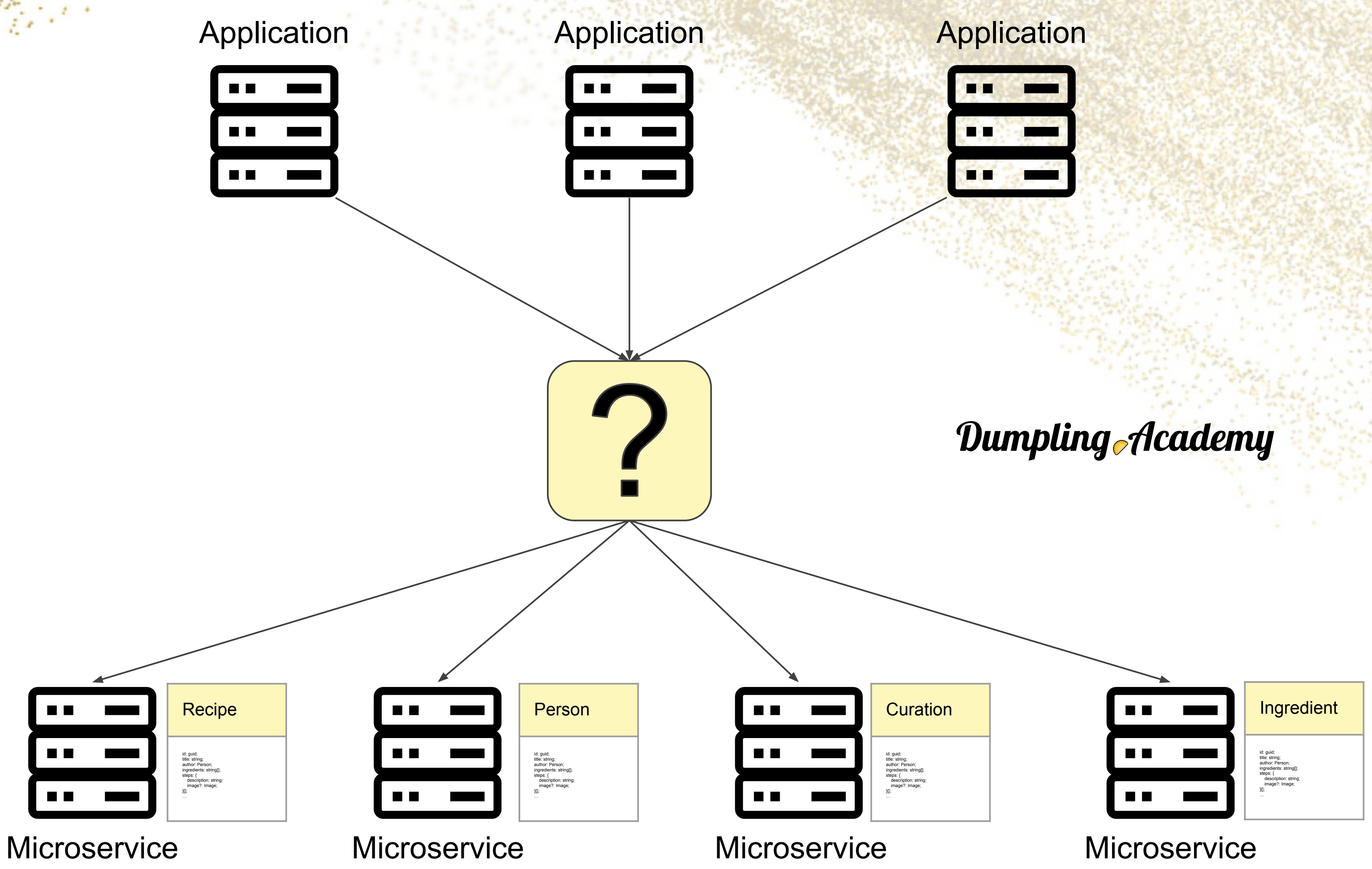


```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
...
```

Dumpling Academy

Factoring in a dedicated layer to selectively expose functionality is an example of the Facade pattern, and this can most certainly help alleviate some of our level 2 pain points! While we are still adding another service, and taking on the organizational and infrastructural costs associated with doing so, a facade service can help reduce the number of updates needed whenever a microservice updates, as well as add back a small amount of consistency in that each application only needs to manage the ability to talk to the facade.

Level 2 / Level 3



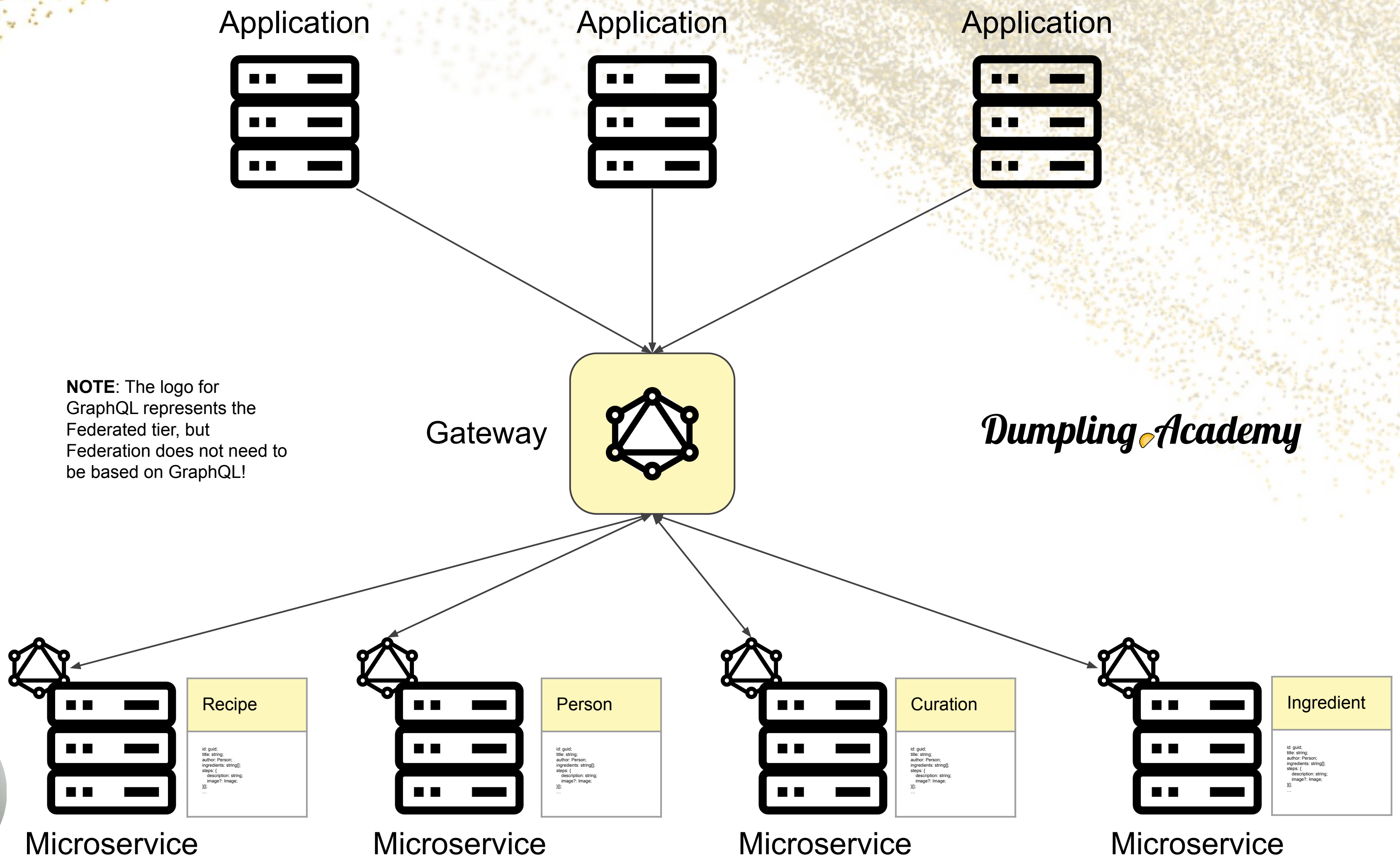
```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
...
```

Dumpling Academy

So what would our level 3 chef do differently? While they acknowledge that reducing the pain of having to replicate data changes to every consumer application is a recipe for success, there's room for improvement. Likewise a pure facade will only be exposing access to the various microservices without adding logic itself, meaning multiple calls to the facade. You could potentially add the ability for the facade to handle one-to-many calls, but that re-introduces violations of logical isolation.

The answer comes from a very similar place as our earlier level progression, added constraints! From level 1 to level 2, we added organizational constraints based on the various types of data and how they were used. This allowed us to create individual services for each type, and grow the overall system accordingly. The same is true from level 2 to level 3, except instead of an organizational constraint on the type of data, we're going to introduce one on the shape of the data, through an approach known as Federation.

Level 3: Federated Gateway



Recipe

```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
...
```

Person

```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
...
```

Curation

```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
...
```

Ingredient

```
id: guid;
title: string;
author: Person;
ingredients: string[];
steps: {
   description: string;
   image?: Image;
}[];
...
```

NOTE: The logo for GraphQL represents the Federated tier, but Federation does not need to be based on GraphQL!

Dumpling Academy

Federation operates under the same compartmentalization paradigm as our earlier microservices approach, but additionally provides strict guidance for the facade tier, which in this pattern is called the Gateway. Because the shape of your data is well defined coming from each sub-service, the gateway is able to act more intelligently than a traditional facade. If an application asks the gateway for recipe data containing fields related to the author, the gateway is able to leverage the known shape of data and interpret that into a call to both the recipe and person services, and return all desired information in a single request.

The Federated approach also defines a process called “introspection”, which allows the gateway to request the current shape of the data each dependency is able to provide. This enables the gateway to dynamically compose itself without direct code changes or deployments, meaning that the only time you want to actively modify the gateway is when you want to add new microservices behind it!

Compare & Contrast: Facade vs. Federation

- Both patterns leverage creating a common access tier for consumers to interact with multiple dependent systems, without needing full knowledge of the details of those systems. The Federated approach has an additional constraint on the shape of the data called the federation protocol

- A pure facade only helps with routing to services and slightly fewer updates. A facade service can do more, but requires workflow-based logic in the facade itself, violating the goal for logical isolation

- A service leveraging federation has the same benefits as a facade service, but doesn't require defining logic in the gateway, due to the federation protocol

- You can utilize introspection to dynamically retrieve protocol updates from connected microserves, further reducing development burden around updates

Level 2 / Level 3

Both patterns help reduce repetition with regard to contractual updates, and require different trade-offs.

A facade can be pure and contain no workflow logic, or a microservice itself that needs to be managed properly by any teams owning those flows.

The federated gateway gets the benefit of each facade variant; being updated only with new service like a pure facade, but able to merge requests between subservices like a facade service. This is due to strict conformity to the federation protocol, which also yields the benefit of introspection.

It's important to fully evaluate what it means to conform to a federation protocol before jumping in, because the cost of adopting such an approach may be high. If your services are already leveraging well-defined communication form, such as with GraphQL, it may be easier to adopt federation if you so desire.

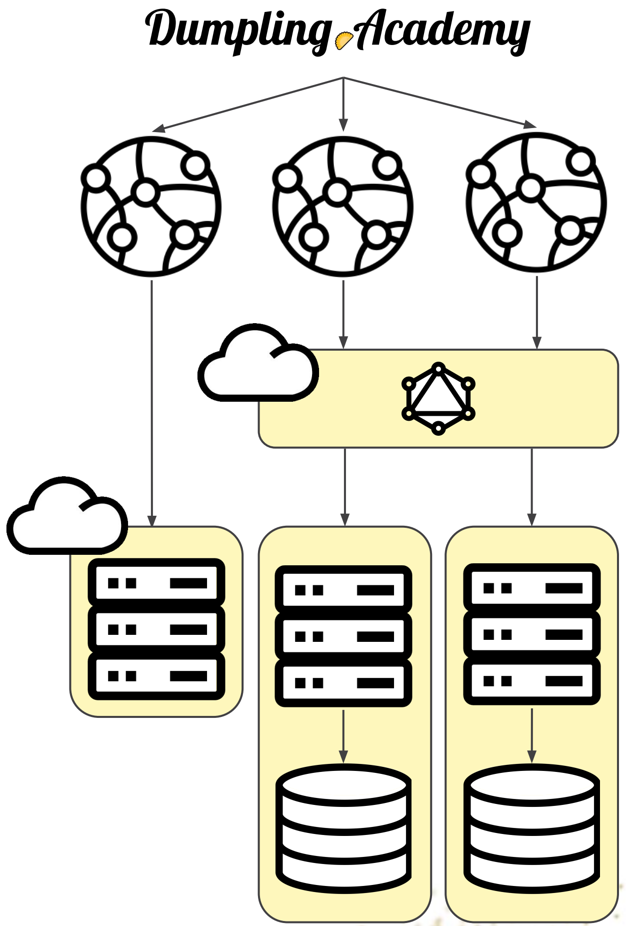
Level 3: Federation

Pros

- **Developer Experience**
  - Deployment
  - Development
- **Uniformity**
  - Standardization
- **Management**
  - Ownership
- **Performance**
  - Reliability
- **Logical Isolation**
- **Growth**

Cons

- **Developer Experience**
  - Debugging
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Management**
  - Organizational Overhead
  - Ownership
- **Infrastructure**
  - Cost
  - Technical Overhead
- **Networking**
  - Latency



database icon by zwicon on svgrepo.com

So what exactly does federation offer Dumpling Academy? First and foremost, it still provides the same benefits to developer experience as the microservices approach. The federation protocol provides standardization to communication that applies even if some services are written in a different language.

Each microservice can still be owned separately and updated freely, which continues to benefit the stability of releases and how well the services perform. The gateway and protocol allow for continued logical isolation as all data definitions live with each microservice, even if they are accessible elsewhere in the system. Updates to existing microservices are easier to make, and new microservices can be added at any time, just as before.

Level 3: Federation

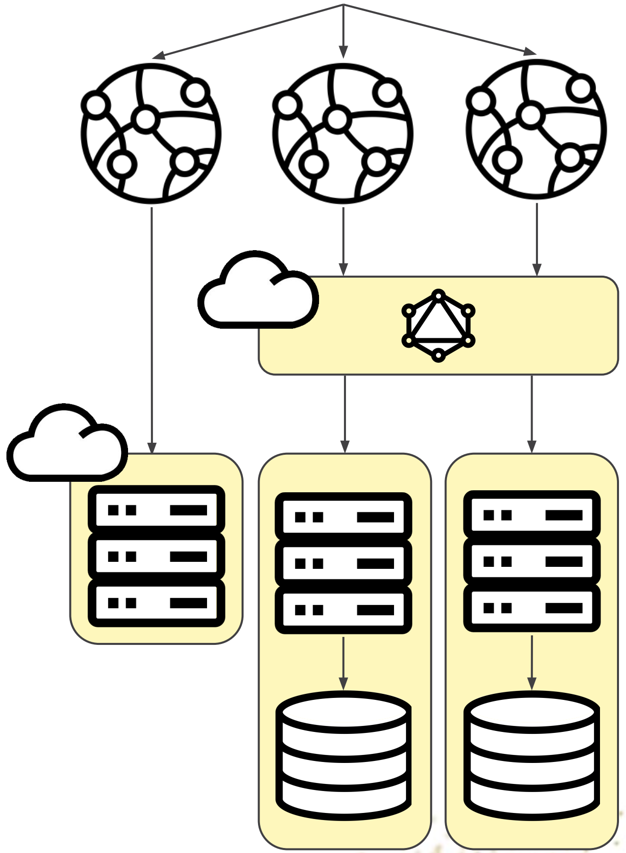
Pros

- **Developer Experience**
  - Deployment
  - Development
- **Uniformity**
  - Standardization
- **Management**
  - Ownership
- **Performance**
  - Reliability
- **Logical Isolation**
- **Growth**

Cons

- **Developer Experience**
  - Debugging
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Management**
  - Organizational Overhead
  - Ownership
- **Infrastructure**
  - Cost
  - Technical Overhead
- **Networking**
  - Latency

Dumpling Academy



database icon by zwicon on svgrepo.com

Similarly, the drawbacks to this approach are very similar to what we saw with traditional microservices, where integration between systems is less straightforward. While the federation protocol provides many benefits, there is a cost to keeping everything compliant, and if you're adopting for the first time, it can be a large effort to bring existing systems into compliance.

The gateway tier can operate without explicit workflow logic, but interacts with every workflow, so what team maintains it? Furthermore, every team using the gateway now has a knowledge dependency on how it works. The gateway tier brings on at least one additional service to pay for and manage, and likely with higher throughput compared to the other services in the system due to its role. And while the federated approach is ideally reducing the number of network round-trips between services, there are still latency concerns as well.

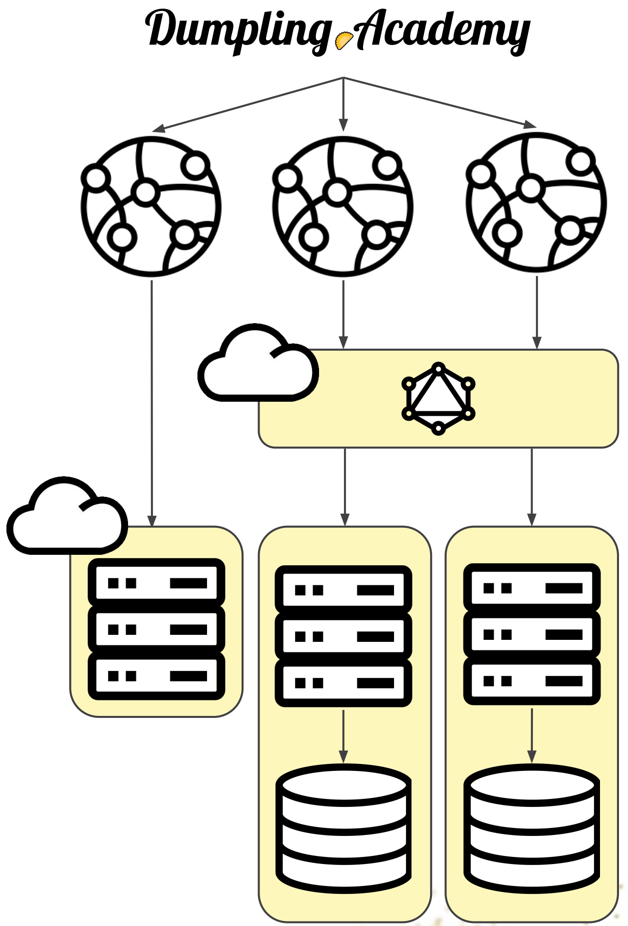
Level 3: Federation

Pros

- **Developer Experience**
  - Deployment
  - Development
- **Uniformity**
  - Standardization
- **Management**
  - Ownership
- **Performance**
  - Reliability
- **Logical Isolation**
- **Growth**

Cons

- **Developer Experience**
  - Debugging
  - Testing
- **Uniformity**
  - Consistency
  - Standardization
- **Management**
  - Organizational Overhead
  - Ownership
- **Infrastructure**
  - Cost
  - Technical Overhead
- **Networking**
  - Latency



The federated approach relies on strict organizational and data-structural constraints in order to provide additional benefit beyond a traditional microservices approach. This deliberate trade-off of increased upfront complexity for reduced time and effort later, can certainly provide benefits!

But it can be easy to make this trade-off and then find yourself in a situation where it isn’t worth it, and before we talk about possible situations where this may happen, it’s finally time to meet our food scientist.

Level 4: Food Scientist

Now in my mind level 4 is a bit of a misnomer here, because the “Food Scientist” isn’t producing any dumplings themselves. Instead their role is to highlight what each chef has done well during the preparation of their dish, investigate anything that went wrong, and provide context for the strengths and shortcomings of each approach, as well as where techniques can be shared between them.

So let's begin by reviewing the mindset of each chef when making dumplings.

Level 1: Amateur

- Holistic concerns

Level 2: Home Chef

- Compartmentalized concerns
- Incurs organizational overhead to the benefit of logical isolation of concerns

Level 3: Professional

- Adaptive concerns
- Incurs organizational overhead to the benefit of logical isolation of concerns
- Incurs structural overhead for further benefit

The level one chef is focused on holistic concerns around the creation of their dumpling and they'll use the equipment and supplies that enable them to make the best possible dumpling they can.

The level two chef leverages "mise en place" as a technique for compartmentalizing concerns and ensuring that each part of their dumpling is given individual focus. They proactively invest effort into more strictly organizing their process and sub-products before starting to be able to handle more concerns along the way to their ideal dumpling.

And finally the level three chef, who also leverages strict organization around the cooking process, but builds on that with structure of the kitchen, which is needed as they generally are producing more components for a wider variety of dumplings.

While each chef in the "Four Levels" series is differentiated by experience, the level 4 mindset acknowledges that it isn't their defining characteristic.

Level 4

	Level 1: Monolith	Level 2: Microservices	Level 3: Federation
When to Use	• Consistent product unlikely to change • Budgetary, Developmental, or Infrastructural limitations	• Ability to leverage existing systems • Non-overlapping or heavily customized concerns • Parallelizable work	• An existing Microservice structure is insufficient in addressing business concerns • All Data adheres to a federation protocol
When to Change	• Scope has diversified from original designs • Singular codebase is difficult to manage for developers	• Substantial overhead managing connection to a large number of microservices • Additional logistical and infrastructural costs are palatable	• Access patterns don't benefit from federation • Upkeep and Overhead no longer beneficial

While we are still calling each approach a "level", that doesn't mean any given approach is better than another in all cases, there are actually places where each pattern excels, and well as where they don't.

A monolith is a great choice for a product with consistency and uniformity. It can also be a choice by necessity, where constraints such as budget or resourcing mean that managing multiple services on a day-to-day basis just isn't feasible. And yes, a monolith is a common pattern when just starting out, but I'd like to argue that's because the concerns when starting are generally smaller and more related, and not because it's an easier or less skilled pattern.

If you have a monolith, recall some of the pain points we discussed earlier: realizing that your product is no longer as well-defined or manageable given current resources as it once was. Splitting the monolith to microservices is not a small task, but if you can align on an organization that benefits your approaches, it may be worth it.

Level 4

	Level 1: Monolith	Level 2: Microservices	Level 3: Federation
When to Use	• Consistent product unlikely to change • Budgetary, Developmental, or Infrastructural limitations	• Ability to leverage existing systems • Non-overlapping or heavily customized concerns • Parallelizable work	• An existing Microservice structure is insufficient in addressing business concerns • All Data adheres to a federation protocol
When to Change	• Scope has diversified from original designs • Singular codebase is difficult to manage for developers	• Substantial overhead managing connection to a large number of microservices • Additional logistical and infrastructural costs are palatable	• Access patterns don't benefit from federation • Upkeep and Overhead no longer beneficial

Leveraging the various benefits of proper organization is the biggest benefit of microservices. If services already exist you can add more quite easily or build on top of existing ones, though doing so creates some concerns about latency if you aren't careful. The approach is significantly better for parallelizing work among multiple teams, since separating developmental concerns is the foundational assumption.

Not everyone who adopts microservices is going to find the pattern benefits them enough where the additional costs and overhead is worth it, and that that's ok. While it isn't the easiest task to switch back and forth between the patterns, it may better support the needs of your product to switch at times. But if some additional cost is palatable, there are additional factors that can be addressed by transitioning a microservices architecture into a federated one.

Level 4

	Level 1: Monolith	Level 2: Microservices	Level 3: Federation
When to Use	• Consistent product unlikely to change • Budgetary, Developmental, or Infrastructural limitations	• Ability to leverage existing systems • Non-overlapping or heavily customized concerns • Parallelizable work	• An existing Microservice structure is insufficient in addressing business concerns • All Data adheres to a federation protocol
When to Change	• Scope has diversified from original designs • Singular codebase is difficult to manage for developers	• Substantial overhead managing connection to a large number of microservices • Additional logistical and infrastructural costs are palatable	• Access patterns don't benefit from federation • Upkeep and Overhead no longer beneficial

If your microservices were already close to adhering to a federation protocol, which is a very real possibility, the switch may not be significantly more than the lift for another new microservice. And many products are able to comply with strict structural standards for data, making the developmental benefits of the federated gateway a compelling choice! But it's also important to acknowledge the opposite, that there are some products and usage patterns that wouldn't benefit from a federated gateway, and if you do have such a product, don't force it! Federation is an investment towards developmental growth, and if that outlook ever does change, or the management of a shared tier isn't working out, don't be afraid to move back towards traditional microservices either.

	Level 1: Monolith	Level 2: Microservices	Level 3: Federation
When to Use	● Consistent product unlikely to change ● Budgetary, Developmental, or Infrastructural limitations	● Ability to leverage existing systems ● Non-overlapping or heavily customized concerns ● Parallelizable work	● An existing Microservice structure is insufficient in addressing business concerns ● All Data adheres to a federation protocol
When to Change	● Scope has diversified from original designs ● Singular codebase is difficult to manage for developers	● Substantial overhead managing connection to a large number of microservices ● Additional logistical and infrastructural costs are palatable	● Access patterns don't benefit from federation ● Upkeep and Overhead no longer beneficial

Level 4

Each approach is the right choice for a certain scenario, and it's even possible to start at the beginning with any of them. In fact, if you have well-defined data from the get-go, federation is a fantastic choice! But in general consider starting with either a monolith or traditional microservices structure depending on available resources and initial expectations.

And so we've reached the end of our comparisons, and have hopefully reinforced the value of each approach in the right situation. As we wrap up, I'd like to ask you to consider a few closing items. First…

Level 4

Where are you now?

Which level is right for you?

Think of a product you're involved with, and where you are now with regard to the current approach you follow. How well it is working for you?

Second, for that same product, which level is right for you? If it's where you already are, do you need to make any refinements to ensure everything continues smoothly? If it isn't, how can you transition to the right level?

Third…

Consider: D.U.M.P.L.I.N.Gs

Developer Experience	Debugging, Deployment, Development, Testing
Uniformity	Consistency, Standardization
Management	Organizational Overhead, Ownership
Performance	Reliability, Speed
Logical Isolation	
Infrastructure	Cost, Technical Overhead
Networking	Latency, Load
Growth	

Level 4

…let's consider Dumplings, but finally tying it to our technology discussion. What aspects of our dumplings do we most care about?

Of all the different pros and cons discussed today, which is most important to you and where are you willing to make trade-offs? Which approach best emphasizes that balance?

Regardless of which "level" you're at and where you want to be, so long as you identify and work towards the aspects you care about, I bet you'll be happy with your dumplings.

4 LEVELS

Bloomberg

Engineering

4 levels logo referencing the Epicurious series created from pierogi line art by Lucas Collins on pngitem.com.

And finally, leaving our framework and food analogies behind, I'd like to offer anyone who made it this far some free access to bloomberg.com; Check out our news content, and perhaps observe a product that was able to leverage some aspects of our discussion that weren't so hypothetical after all.

Thank you.

—

Reference: dumpling.academy