Microservice Architecture for the Enterprise

How to take a design-based approach to microservice architecture that addresses culture, organization, methodology and technology.

A Broadcom Company

What are microservices?

For the uninitiated, a microservice is defined as an independently deployable component of bounded scope that supports interoperability through message-based communication.

But, it's not just about individual microservices—it's about how microservices work together. From this perspective, we look at "microservice architecture"—a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices. This definition comes from the book *Microservice Architecture*, published by O'Reilly.

What does this mean for businesses? Rather than building monolithic applications that require long development cycles and big releases, organizations are creating applications out of multiple, lightweight microservices that facilitate smaller, more frequent changes and independent scalability.

In the pages that follow, we'll cover how your organization can **improve availability and system safety** while **speeding up and scaling software delivery** using a microservice architecture.

Why microservices in the enterprise?

Companies like Amazon and Netflix popularized the use of microservices, and other pioneers—such as Gilt (now part of HBC Digital) and SoundCloud—use them to build massive scalability and decrease time between releases.

Netflix started using a service-oriented approach to software engineering that we now call microservices over 10 years ago. In a 2006 interview, Amazon CTO, Werner Vogels, explained the benefits:

"We can scale our operation independently, maintain unparalleled system availability and introduce new services quickly without the need for massive reconfiguration."

Gilt uses microservices to lessen dependencies between teams, allowing the company to get code more quickly into production. SoundCloud, after expanding as a company, progressed from using agile to continuous delivery to microservices to improve lead and delivery times that it was accustomed to in its start up days.

While such companies are digital first, **large enterprises with legacy systems to support can also benefit from a microservice architecture.** And though the challenges—such as monolithic systems, legacy technology, skills gaps and cultural issues—might be great, the rewards can be far greater.

What are the challenges?

From an enterprise perspective, much of what we're dealing with when introducing microservices and trying to get advantages from microservices is the inevitability of software complexity. And software complexity is certainly a subject that's been studied much longer than we've been talking about the buzz term "microservices." Software complexity goes back almost to the origins of software engineering.

Fred Brooks describes two types of complexity in his paper "No Silver Bullet: Essence and Accident in Software Engineering":

- **Essential complexity**—The complexity of the software's functional scope and the problems it solves (e.g., correlating and analyzing large amounts of data in real time). There are some problems we are trying to solve with software that are inherently complex and we can't do anything about it.

- **Accidental complexity**—The complexity of the software's implementation details (e.g., the languages, processes and messages used to do the work). This is the complexity we create in trying to solve the problem.

The difference between these is that you'll never get rid of the essential complexity. In fact, software solutions are a great way to try to deal with essential complexity. On the other hand, accidental complexity is something that we would hope to reduce as much as possible.

How do we overcome the challenges of building an enterprise microservice architecture?

So, how do we deal with **essential complexity**? There's a reason domain-driven design has become resurgent with the microservices movement. Domain-driven design, at a high level, is a very effective way of modeling the systems which then map nicely into microservice architectures.

This is possible because the topology of the implemented system closely resembles the model of the system's "essence." In other words, there's a close resemblance between a model of the essential complexity of a system and a model of a microservice architecture. It makes microservice architecture an intuitive way of solving these essential complexity problems.

On the other hand, **accidental complexity** in a microservice architecture can be minimized through automation and distribution via continuous delivery tooling, cloud native platforms, containers and APIs.

What we see with microservices is now, instead of the developers building their code and having to merge it with a monolithic application, they can **break things down and work on the individual microservices,** eliminating a lot of the accidental complexity. Their job gets simpler and they can just focus on building a service.

A Design-Based Approach to Microservice Architecture

When looking at how to approach the introduction of microservices to an enterprise, taking a design-based approach is very helpful. This approach can be broken down into **five different steps** of design:

1

Outcome design. Look at your goals and ask, "Why are you doing this?" It's not enough to say: "Everybody's on the microservice bandwagon, let's jump on too." It's important that you understand the value points you're going after.

2

System design. Examine how you identify the scope of the system that you're going to be architecting. This is about decomposing the domain.

3

Service design. Once you have a picture of what your domain is and what all its services are going to be, look at the design of all those individual services to make sure they're built in the right way, so that they can evolve and interact in the correct way for the system you want to build.

4

Foundation design. The previous steps have been very technology-agnostic. So now, you need to look at the underlying capabilities—the technological tools and platforms that will be required to build out the system best-suited to the needs of your organization.

5

Organizational design. Look at the organization itself—the people side. How do you make sure that the culture and methodologies you're using, and even the organizational structure, match what you hope to achieve?

We'll dig into these a little deeper in the pages that follow.

1 Outcome Design: Define Goals and Principles

STEP 1 is about **defining measurable goals** and **developing associated principles.** First, you want to define where you are today and where you want to be. Ask:

- How can we reduce our release cycles?
- How can we introduce more microservices?
- How can we retire or depreciate unused services that may be out there?

Naturally, not everything can be concrete and measurable. Sometimes, it's important to define higher-level principles about the way things should be done to incent good behavior. Defining these principles helps you build a common cause, which is very important. We want this so that we can have alignment without having to have a lot of coordination across the organization. So, these are constraints to help you with those hard-to-measure goals.

There are some key types of microservice goals such as **agility, composability, runtime and scalability.** Out of these, you can decide what are the principles you want to use to incent these behaviors.

2 System Design: Choose an Initial Scope Then Decompose the Domain

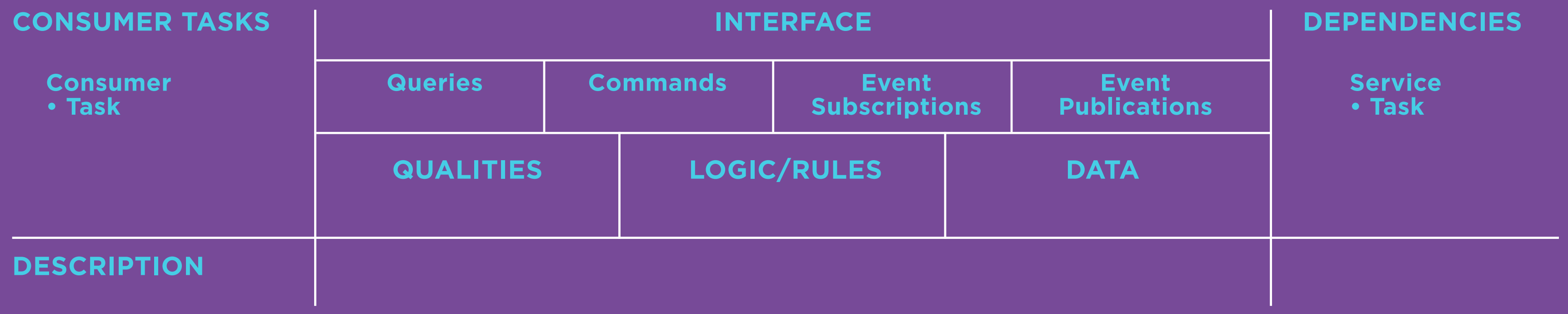
Once you've defined your goals and principles, look at the best place to start working towards achieving them. In **STEP 2**, you want to first **identify the target domain** and **start breaking that down into sub-domains in a bounded context**—which are linguistic boundaries where everyone speaks the language. Then, start to define the interaction between those bounded contexts within your domain.

This is a process that's helped through visual exercise. One approach is to use the domain-driven design "Context Mapping" approach that breaks down a domain into its sub-domains and bounded contexts. Since the accepted approach to understanding a system is to focus on the relationship between its components, you need not go deeper than the context map if you want a basic representation of a microservice system.

This context could be the decomposition of a monolithic application or the service interactions of an initiative. The only way to coherently build a large organization's system of microservices is to do so piece by piece, context by context. Along the way, these contexts can be combined to project the complete picture, if such a picture is even needed.

Once you have the bounded contexts, you can start to define services for your solution within. And once you've got a good map, you can start enumerating the services that you want in the organization.

RELATED READING
Designing a System of Microservices



3 Service Design: Design the Services

In **STEP 3,** you want to **take an outside-in approach**. Start with:

- What are the interactions between services?
- Who is going to want to consume my services and what are they going to consume?
- What other services am I going to be dependent on?

Then, you start to think about other external concerns like the quality of service, SLAs, the security situation and versioning—all the things that are going to affect the consuming services as well as the services you depend on. These are going to be the most important factors in designing the service.

From there, you should start to investigate the logic, the rules and finally, the data you need. It's a common mistake to start with the data and work outwards. That's dangerous and it leads to tight coupling.

RELATED READING
The Microservice Design Canvas

You can expect the stages of system design and service design to be quite iterative at first until things settle down and the boundaries become clear. But, it's important to remember that the system design is going to be enterprise-wide at the system maintainer level and the service design is going to be much closer to the individual developers and the small teams that will be building the individual microservices.

3 Service Design (cont'd): The Importance of APIs

Another thing to think about when you're designing the services is to dig into the APIs because they're extremely helpful and have a lot of value when it comes to microservices. APIs provide a technological way of **expressing the capabilities of your system.**

If you didn't know anything about a system and you were only to look at the API definition for the services that live within the system, you would get a pretty good idea of what functional capability is in that system to the point where you could have business and technical conversations, bridging that gap using the API definitions.

In addition, APIs are:

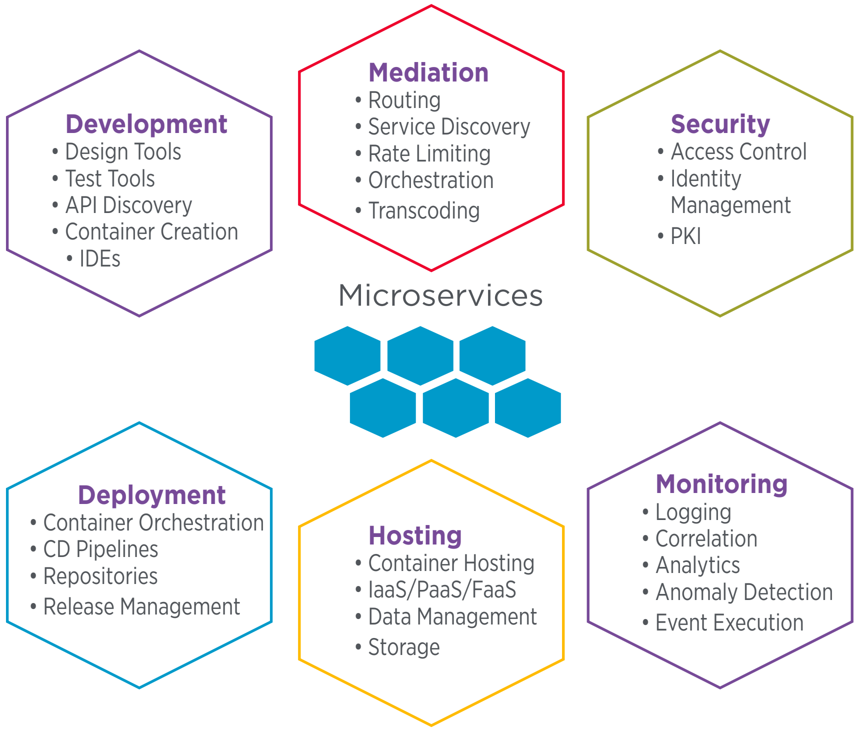
- A living part of the system, unlike some documentation that lives outside the system and might be stale.
- A good way of bridging to service modelling through domain-driven design. As you go from domain to bounded context to the context map, think about your APIs as a helpful way to map your system. It provides a way to understand the overall system of microservices, and it contains the appropriate level of information so you can connect the dots.
- A place where you can enforce security policies, provide composition of services for aggregation, universally monitor service levels and document services through API definition languages, like OpenAPI. This is important because when you have a microservice, you can no longer rely on something like an app server to provide all the normalized information, such as logging and security.

4 Foundation Design: Identify Needed Capabilities

Now it's time to **think about the required capabilities**. These could be technical capabilities or they could be standards and guidelines you want to abide by. But, be careful not to go so deep as to be restrictive to your teams.

To assist, use this capability model that outlines some of the main categories you may want to consider. Not to say every microservice environment needs all these capabilities, but you can use the model to think about the important capabilities you need in your system.

If you're focused on reducing development time, you may spend more time looking at design and development capabilities. Whereas if your focus is more on scalability and runtime efficiency, you may be looking more deeply into security and mediation and platform capabilities.



5 Organizational Design: Align the Organization, Culture and Architecture

Now we get into the human side of microservices: the organizational, cultural and methodological capabilities. When introducing something like continuous delivery, it's more than just a technology tool introduction. It's the whole process change that impacts the way people do things. And it's important to **align your team structure** so you have cross-functional teams that can own their services end-to-end. Also, you want to look at how you can change culture.

On the **methodological side**, you'll want to adopt agile practices. You want to be able to automate everything.

On the **organizational side**, you want to break down silos into teams that are cross-functional—product owners and developers, and even business leads in the same team—so you can be more aligned with the business outcomes. There will still be the need for teams that go across the organization and support those cross-functional teams, but rather than dictating to those teams, it's better to take a more incentive-based approach to teams where you're enabling those teams and providing tools and services to them, to let them function and fly on their own.

Finally, on the **cultural side**, if you have digital teams that are counterpoint to their IT teams, it's time for a change. While change can be viewed as bad, which slows down releases, try thinking about becoming efficient at change and embracing it, so that you're doing more frequent, smaller changes, which are going to have less impact on the system and are much easier to roll back if things do go wrong.

Move to Microservices and Accelerate Your Digital Transformation

You want to deliver new innovations, release apps faster and take advantage of new opportunities, but legacy applications and infrastructure are holding you back. Transition to a modern architecture by decomposing monolithic applications into agile microservices—independently created, managed and scaled. Your business will be able to act more quickly and developers will love the easy access to APIs that give them the freedom to focus on customer experience.

Start With Microservice Strategy and Design

CA,a Broadcom Company has the API Academy, an educational site where you can learn about API strategy and design, as well as microservice architecture and the education enterprise architects need to build better APIs and microservices, improve software delivery, and execute on broader digital strategies.

Read API Academy Microservice Best Practices

Low-Code Microservice Creation

Live API Creator is the only automated, low-code microservices development solution and works up to 10 times faster than other approaches. It creates and exposes domain-driven microservices and REST APIs as application backends, providing access to orchestrated data and functionality from both new and legacy systems.

Learn more about Live API Creator

Orchestrate and Secure Microservices

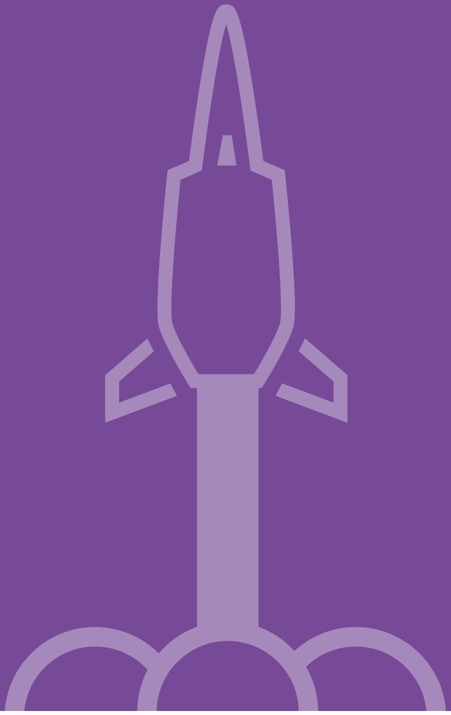
The award-winning API Gateway and Microgateway enable architects and developers to manage discovery, orchestration and transformation in a broad array of microservices deployment patterns. They are containerized and deployable in Docker® and you'll also be able to apply best-in-class OAuth security and authentication to protect your business.

Learn more about API Gateway & Microgateway

Application Architecture Built for Change

Start building microservices today with a trial of Live API Creator. Get started at ca.com/createapis.

Learn how Layer7 can help you with your microservice architecture. Visit ca.com/microservices.



For product information please visit us at ca.com/api

ca technologies A Broadcom Company