



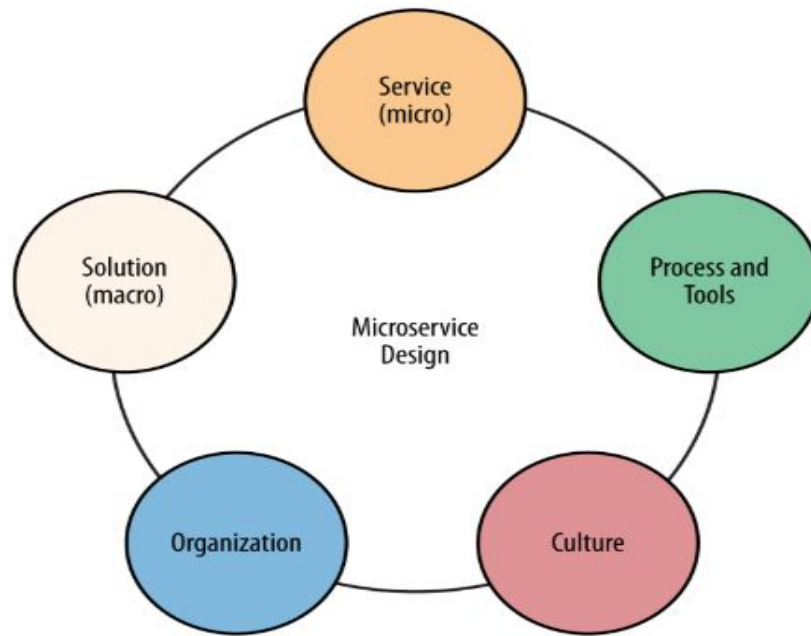
Unit 01

Chapter 3

The Systems Approach to Microservices

A microservices system encompasses all of the things about your organization that are related to the application it produces.

This means that the structure of your organization, the people who work there, the way they work, and the outputs they produce are all important system factors. Equally important are runtime architectural elements such as service coordination, error handling, and operational practices.



The microservice system design model

Microservice Design Model

SERVICE

In a microservice system, the services form the atomic building blocks from which the entire organism is built.

SOLUTION

When designing a particular microservice your decisions are bounded by the need to produce a single output—the service itself. Conversely, when designing a solution architecture your decisions are bounded by the need to coordinate all the inputs and outputs of multiple services.

PROCESS AND TOOLS

Choosing the right processes and tools is an important factor in producing good microservice system behavior. For example, adopting standardized processes like DevOps and Agile or tools like Docker containers can increase the changeability of your system.

Tools and processes related to software development, code deployment, maintenance, and product management.

Continue...

Organization

From a microservice system perspective, organizational design includes the structure, direction of authority, granularity, and composition of teams.

A good service design is a byproduct of good organizational design.

Culture

We can broadly define culture as a set of values, beliefs, or ideals that are shared by all of the workers within an organization. Your organization's culture is important because it shapes all of the atomic decisions that people within the system will make.

Much like organizational design, culture is a context-sensitive feature of your system.

All the elements are interconnected and a change to one element can have a meaningful and sometimes unpredictable impact on other elements.

Standardization and Coordination

Mastering the system you are designing and making it do the things you want requires you to develop the right standards, make sure the standards are being applied, and measure the results of the changes you are making. Standardization is the enemy of adaptability and if you standardize too many parts of your system you risk creating something that is costly and difficult to change.(example of farmer crops)

Standardizing process

- Standardizing how we work has broad-reaching implications on the type of work we can produce, the kind of people we hire, and the culture of an organization.
- Agile institutionalizes the concept that change should be introduced in small measurable increments that allow the organization to handle change easier.
- Teams to define the types of tools their workers are allowed to utilize. For example, some firms forbid the use of open source software and limit their teams to the use of centrally approved software, procured by a specialist team.

Continue...

Standardizing outputs

Output standardization is way of setting a universal standard for what that output should look like.

In a microservices system, a team takes a set of requirements and turns those into a microservice. So, the service is the output and the face of that output is the interface (or API) that provides access to the features and data the microservice provides.

For example,

You might decide that all the organization's services should have an HTTP interface or that all services should be capable of subscribing to and emitting events.

For example, in an assembly line the output of the line workers is standardized—everyone on the line must produce exactly the same result. Any deviation from the standard output is considered a failure.

Continue...

Standardizing people

Standardizing skills or talent can be an effective way of introducing more autonomy into your microservices system.

For example, you could introduce a minimum skill requirement for anyone who wants to work on a microservice team.

All organizations have some level of minimum skill and experience level for their workers, but organizations that prioritize skill standardization often set very high specialist requirements in order to reap system benefits. If only the best and brightest are good enough to work within your system, be prepared to pay a high cost to maintain that standard.

Continue...

Standardization trade-offs

Standardization helps you exert influence over your system, but you don't have to choose just one of these standards to utilize. But keep in mind that while they aren't mutually exclusive, the introduction of different modes of standardization can create unintended consequences in other parts of the system.

For Example, you might decide to standardize on the APIs that all microservices expose because you want to reduce the cost of connecting things together in your solution architecture. To do this you might prescribe a set of rules for the types of APIs that developers are allowed to create and institute a review process to police this standardization. As an example, many organizations standardize a way of documenting the interfaces that are created. At the moment Swagger (also called OpenAPI) is a popular example of an interface description language,

A Microservice Design Process

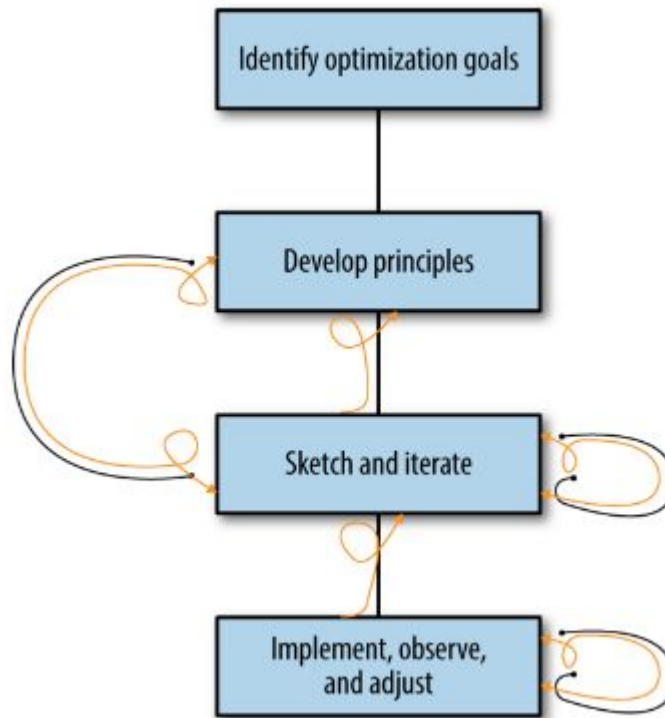
A good designer employs a process that helps them continually get closer to the best product.

Choose the best design that help you understand the the impact of your assumptions and the applicability of advice as you change the system.

SET OPTIMIZATION GOALS

The behavior of your microservice system is “correct” when it helps you achieve your goals.

For example, a financial information system might be optimized for reliability and security above all other factors. That doesn’t mean that changeability, usability, and other system qualities are unimportant—it simply means that the designers will always make decisions that favor security and reliability above all other things.



A framework for Microservice system design process

Development Principles

Principles outline the general policies, constraints, and ideals that should be applied universally to the actors within the system to guide decision-making and behavior. The best designed principles are simply stated, easy to understand, and have a profound impact on the system they act upon.

Sketch the System Design

A good approach is to sketch the important parts of your system design for the purposes of evaluation and iteration.

The goal of a sketching exercise is to continually improve the design until you are comfortable moving forward.

That is to sketch out the core parts of your system, including **organizational structure**, (how big are the teams? what is the direction of authority? who is on the team?), **the solution architecture** (how are services organized? what infrastructure must be in place?), **the service design** (what outputs? how big?), and **the processes and tools** (how do services get deployed? what tools are necessary?). You should evaluate these decisions against the goals and principles you've outlined earlier. Will your system foster those goals? Do the principles make sense? Do the principles need to change? Does the system design need to change?

Good sketches are easy to make and easy to destroy, so avoid modeling your system in a way that requires a heavy investment of time or effort. The more effort it takes to sketch your system the less likely you are to throw it away.

Implement, Observe, and Adjust

- Bad designers make assumptions about how a system works, apply changes in the hope that it will produce desired behavior, and call it a day. Good designers make small system changes, assess the impact of those changes, and continually prod the system behavior toward a desired outcome.
- key performance indicator (KPI) - gain essential visibility into our system by identifying a few key measurements that give us the most valuable information about system behavior.
- Designers working on existing applications can observe the existing and past behavior of the system to identify patterns.
- Designers who are working on new applications often have very little information to start with—the only way to identify the brittle points of the application is to ship the product and see what happens.
- Cost of changing microservice is high because you need special permission, additional funds, more people, or more time to make the changes you want to the system.
- So, in order to design a microservice system that is dynamic you'll need to identify the right KPIs, be able to interpret the data, and make small, cheap changes to the system that can guide you back on the right course. This is only possible if the right organization, culture, processes, and system architecture are in place to make it cheap and easy to do so.

The Microservices System Designer

To be most effective, the microservices system designer should be able to enact change to a wide array of system concerns.

the changes you enact could have a broad-reaching impact. Alternatively, you could focus on a particular team or division within the company and build a system that aligns with the parent company's strategic goals. Introduce new changes within the system to meet with the desired company goal.

The solution architect focuses on the coordination of services, the team manager focuses on the people, and the service developer focuses on the service design. We believe that someone or some team must be responsible for the holistic view of the entire system for a microservices system to succeed.



Unit 01

Chapter 4

Goals and Principles

Goals and principles to help inform your design choices and guide the implementation efforts.

Goals for the Microservices Way

The microservices way: Finding the right harmony of speed and safety at scale.

Here are the four goals to consider:

1. Reduce Cost: Will this reduce overall cost of designing, implementing, and maintaining IT services?
2. Increase Release Speed: Will this increase the speed at which my team can get from idea to deployment of services?
3. Improve Resilience: Will this improve the resilience of our service network?
4. Enable Visibility: Does this help me better see what is going on in my service network?

Goals

Reduce Cost

- The ability to reduce the cost of designing, implementing, and deploying services allows you more flexibility when deciding whether to create a service at all.
- In the operations world, reducing costs was achieved by virtualizing hardware.
- For microservices, this means coming up with ways to reduce the cost of coding and connecting services together.

Continue...


Increase release speed

- Increasing the speed of the “from design to deploy” cycle.
- Like the goal of reducing costs, the ability to increase speed can also lower the risk for attempting new product ideas or even things as simple as new, more efficient data-handling routines.
- By automating important elements of the deployment cycle,


Continue...

Improve Resilience

- No matter the speed or cost of solutions, it is also important to build systems that can “stand up” to unexpected failures.
- When you have an overall system approach (not just focused on a single component or solution) you can aim for creating resilient systems.
- Automation can be used to increase resiliency. By making testing part of the build process, the tests are constantly run against checked-in code, which increases the chances of finding errors in the code.



There are companies that run what they call end-to-end tests before releasing to production but many companies rely on a practice that Jez Humble calls blue-green deployment. In this case, a new release is placed in production with a small subset of users and, if all goes well during a monitoring phase, more users are routed to the new release until the full user base is on the new release. If any problems are encountered during this phased rollout, the users can all be returned to the previous release until problems are resolved and the process starts again.



Continue...

Enable visibility

- Improve the ability of stakeholders to see and understand what is going on in the system.
- Stakeholders can get the reports on the coding backlog, how many builds were created, the number of bugs in the system versus bug completed,
- Most effort to date has been to log and monitor operation-level metrics (memory, storage, throughput, etc.). However, there are some monitoring tools that can take action when things go badly (e.g., reroute traffic).

Operating Principles

Principles offer more concrete guidance or best practices on how to act in order to achieve those goals.

Let's consider various principles used in Netflix's architecture:

1. Antifragility: enforce architectural principles, induce various kinds of failures, and test our ability to survive them.
2. Immutability: Scale horizontally - each released component is immutable, a new version of the service is introduced alongside the old version, on new instances, then traffic is redirected from old to new. After waiting to be sure all is well, the old instances are terminated.
3. Separation of Concerns: Each team owns a group of services. They own building, operating, and evolving those services, and present a stable agreed interface and service level agreement to the consumers of those services.

Continue...

Unix Principles

1. Make each program do one thing well.
2. Expect the output of every program to become the input to another, as yet unknown, program.
3. Design and build software - Don't hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task

Continue...

Suggested Principles

1. Do one thing well
2. Build afresh
3. Expect output to become input
4. Don't insist on interactive input - Reducing the dependency on human interaction in the software development process can go a long way toward increasing the speed at which change occurs.
5. Try early
6. Don't hesitate to throw it away
7. Toolmaking - tools are a means, not an end.

Platforms

A platform with which to make your microservice environment a reality. From a microservice architecture perspective, good platforms increase the harmonic balance between speed and safety of change at scale.

For example, the principle of immutability primarily improves the safety of changes that are made to the system. However, the introduction of containerization tools like Docker make independent deployability easy and greatly reduce the associated costs. When immutability is combined with containerization, both speed and safety of changes are optimized, which may explain the rapid adoption of Docker in large organizations.

Platform for microservice is divided into two groups: shared capabilities and local capabilities

Shared Capabilities

Large enterprises to create a shared set of services for everyone to use.

Shared capabilities are platform services that all teams use. These are standardized things like container technology, policy enforcement, service orchestration/interoperability, and data storage services.

Organizations that highly value safety of changes are more likely to deploy centralized shared capabilities that can offer consistent, predictable results. On the other hand, organizations that desire speed at all costs are likely to avoid shared components as much as possible as it has the potential to inhibit the speed at which decentralized change can be introduced.

The following is a quick rundown of what shared services platforms usually provide:

1. **Hardware services:** placing that completed unit into a rack in the “server room” ready for use by application teams. Another approach is to virtualize the OS and baseline software package as a virtual machine (VM), use of containers
2. **Code management, testing, and deployment:** Once you have running servers as targets, you can deploy application code to them. That’s where code management (e.g., source control and review), testing, and (eventually) deployment come in. For example, the Amazon platform offers automation of testing and deployment that starts as soon a developer checks in her code.
3. **Data stores:**It makes sense for your organization to focus on a select few storage platforms and make those available to all your developer teams.
4. **Service orchestration:** The technology behind service orchestration or service interoperability is another one that is commonly shared across all teams.
5. **Security and identity:** Platform-level security is another shared service. This often happens at the perimeter via gateways and proxies.
6. **Architectural policy:** they are used to enforce company-specific patterns or models—often at runtime through a kind of inspection or even invasive testing.

Local Capabilities

Local capabilities are the ones that are selected and maintained at the team or group level. This allows them to work at their own pace and reduces the number of blocking factors a team will encounter while they work to accomplish their goals.

Here's a rundown of the common local capabilities for microservice environments:

1. **General tooling:** A key local capability is the power to automate the process of rolling out, monitoring, and managing VMs and deployment packages. Netflix created Asgard and Aminator for this. A popular open source tool for this is Jenkins.
2. **Runtime configuration:** A pattern found in many organizations using microservices is the ability roll out new features in a series of controlled stages. This allows teams to assess a new release impact on the rest of the system

3. Service discovery: This ability to abstract the exact location of services allows various teams to make changes to the location of their own service deployments without fear of breaking some other team's existing running code.

4. Request routing: Once you have machines and deployments up and running and discovering services, the actual process of handling requests begins. The simplest form of request routing is just exposing HTTP endpoints from a web server like Apache, Microsoft IIS, NodeJS, and others.

5. System observability: A big challenge in rapidly changing, distributed environments is getting a view of the running instances—seeing their failure/success rates, spotting bottlenecks in the system, etc. There is another class of observability tooling—those that do more than report on system state. These tools actually take action when things seem to be going badly by rerouting traffic, alerting key team members, etc.

Culture

Culture can be described as “shared key values and beliefs” that convey a sense of identity, generate commitment to something larger than the self, and enhances social stability.

Three aspects of culture that you should consider as a foundation for your microservice efforts:

1. Communication: Research shows that the way your teams communicate (both to each other and to other teams) has a direct measurable effect on the quality of your software.

“the very act of organizing a team means certain design decisions have already been made.” The process of deciding things like the size, membership, even the physical location of teams is going to affect the team choices and, ultimately, the team output. Conway”). By considering the communication needs and coordination requirements for a software project, you can set up your teams to make things easier, faster, and to improve overall communication.


2. Team alignment: The size of your teams also has an effect on output. More people on the team means essentially more overhead. Dunbar found that, as groups get larger, more time is spent on maintaining group cohesion. Spotify, the Swedish music streaming company, relies on a team size of around seven

No, communication is terrible!

— Jeff Bezos, Amazon founder and CEO

This led to Bezos' now famous “two-pizza team” rule. Any team that cannot be fed by two pizzas is a team that is too big.

3. Fostering innovation: Innovation can be disruptive to an organization but it is essential to growth and long-term success. The ability to take advantage of creative and innovative ideas is sometimes cited as a reason to adopt a microservice approach to developing software. It's worth noting that being innovative is most often focused on changing something that is already established. This is different than creating something new. Innovation is usually thought of as an opportunity to improve what a team or company already has or is currently doing.



Problem with innovation is that the actual process often looks chaotic from the outside. Innovating can mean coming up with ideas that might not work, that take time to get operating properly, or even start out as more costly and time consuming than the current practice.

Companies we talked to enable innovation by adopting a few key principles. First, they provide a level of autonomy to their teams. They allow teams to determine the best way to handle details within the team. Netflix calls this the principle of “context, not control.”

