MICROSERVICE ORIENTED ARCHITECTURE

THE DEFINITIVE GUIDE: BY PETER MWENDA

MICROSERVICES EXPLAINED

- First, a quick bit of background on how we got here and *why* microservices matters.
- Previously, developers built applications in a way that is now known as the monolith.
- The project starts off small, then something small is added here, a new feature added there and so on.
- In a year or two, you suddenly have this monster of a project where you change one thing and the whole system can break.

MICROSERVICES EXPLAINED …

- Microservice is an architectural design.
- The design structures an application as a collection of services "smaller, more specialized parts".
- These small parts "microservices" communicate with one another across common interfaces such as APIs and REST interfaces like HTTP.

MONOLITHIC APPLICATIONS

- This is a single-tiered software application.
- In most cases, the application comprises of the user interface and data access code. All combined into a single program.
- The three parts of an enterprise application are: a database, client-side user interface (mostly running in a browser), and a server-side application - handles HTTP requests, execute some domain-specific logic, retrieve and update data from the database, and populate the HTML views to be sent to the browser.
- A monolithic application will have at least two of the above in a single executable unit.

MONOLITHIC APPLICATIONS …

- “It's much harder to scale this type of system. It's just one monster project, so you end up having to scale by throwing more servers at it, which ends up being very expensive.”

MICROSERVICES

- The idea with microservices is to focus on building individual services that "does one thing and one thing well".
- Each microservice tends to manage its own database, generate its own logs and handle user authentication. This also usually means that containers are involved at the management and operations level.
- This is a stark contrast to the old monolithic model of building applications as a single entity on a server underpinned by a single relational database.

MICROSERVICES …

- Moving to a more modern approach also tends to bring with it a change in culture for engineering teams, with highly specialized teams giving way to mixed product teams, generally running some flavors of devOps/devSecOps/agile/continuous delivery.

CHALLENGES OF MICROSERVICE

- The complex web of dependencies created by a microservice architecture can lead to some serious headaches.
- First, the biggest issue in changing a monolith into microservices lies in changing the communication pattern.
- Another issue is if the components do not compose cleanly, then all you are doing is shifting complexity from inside a component to the connections between components. This doesn’t just move complexity around, but it also moves it to a place that's less explicit and harder to control.

CHALLENGES OF MICROSERVICE …

- More skillful design is required.
- Breaking a monolith application into microservices may not help if the code is tightly coupled to start with, you will just end up with tightly coupled microservices, which is the worst of all worlds.
- Maintenance can become a pretty big headache to developers “due to poor design”.

PRINCIPLES OF MICROSERVICE ORIENTED ARCHITECTURE

- Architecture is the decisions that you wish you could get right early in a project.
- An architectural style is the combination of distinctive features in which architecture is performed or expressed.
- Recall that, microservice is an architectural style that structures an application as a collection of services. The resulting services should be: Highly maintainable and testable, Loosely coupled, Independently deployable, Organized around business capabilities and Owned by a small team.
- We are going to discus the principles that governs this architecture.

THE PRINCIPLES 1/13

- **Single Responsibility:**

Each microservice must cover and be responsible for a specific feature or functionality or an aggregation of cohesive functionality (single bounded context) encapsulating a functional domain. The rule of thumb in applying this principle is: "Gather together those things that change for the same reason, and separate those things that change for different reasons." — Robert C. Martin.

The Single Responsibility Principle (SRP) is the most foundational principle of good design and states that each software subsystem, module, class, or even function should have one and only one reason to change.

THE PRINCIPLES 2/13

- **Domain Driven Design**

Microservices are organized and built around encapsulated business capabilities. A business capability represents what a business does in a particular domain to fulfill its objectives and responsibilities. Incorporating Domain Driven Design allows the architecture to isolate system ability into various domains. Designing around real-world domains translates to software that represents the real-world problem you are trying to solve.

THE PRINCIPLES 3/13

- **Encapsulation (Black Box)**

A service should only be consumed through a standardized API and should not expose its internal implementation details (composition, business logic, persistence mechanisms e.t.c) to its consumers. Common examples that break the hidden implementation principle include. Team A builds a service that requires its consumers to know its entities database primary key, Services sharing database access e.t.c

THE PRINCIPLES 4/13

- **Location Transparency (Service Discovery)**

Services should never be exposed directly to consumers. Service consumers should never be made aware or dependent on the exact address of the service. The implication of this principle is: A service may change its own url without affecting the Consumer. Services can be deployed to new infrastructure without affecting their consumers. For scaling purposes, services can be deployed multiple times without requiring any reaction from their consumers.

THE PRINCIPLES 5/13

- **Decentralization**

Microservices embrace loosely coupled resources. To achieve autonomy, an organization should strive to push power out of the center, organizationally and architecturally. Each service handling its own responsibility should be favored over having a single orchestrator that holds all business logic in one place. Within a microservices environment there should be no resource centralization.

THE PRINCIPLES 6/13

- **Independently Deployable**

Microservices should be independently deployable as components of the application's ecosystem. This principle enables a change in one service to be deployed without requiring any other services to be deployed.

THE PRINCIPLES 7/13

- **Culture of Automation**

Slicing up a monolithic application into a suite of component services in a highly decomposed ecosystem is bound to result in a noticeable increase in independent deployable units. Embedding automation into the culture of an organization and investing in tooling to support it yields high returns in a microservices environment by dealing with the maintenance overheads an increase in deployable units brings along.

THE PRINCIPLES 8/13

- **Standardized API Mechanism with a Published Contract**

Interfaces should be a well-defined, standardized and widely adopted mechanism that is published and available to its consumers. This API is consistent across other microservices, to encourage re-use and avoid breakages (point-to-point integration). Using a standardized and widely adopted interface mechanism allows the service to be more easily consumed across different systems and languages and by far more consumers in a shorter period of time.

THE PRINCIPLES 9/13

- **Interface Backwards Compatibility**

Interfaces should always remain backwards compatible. Breaking changes to an interface that will force commensurate reactions from a service's consumers should not be permitted.

THE PRINCIPLES 10/13

- **Interface Version Dependency**

Dependency on services should be based on the *interface version* and not on the *version of the application* exposing the interface. Systems should be dependent on the version of the interface they are consuming and completely unaware of the version of the application that exposes it.

THE PRINCIPLES 11/13

- **Resiliency and Failure Isolation**

Microservices architecture is not a silver bullet. It does not automatically make your systems more stable. Considerable effort is required to isolate system failures within the services they occur in to avoid burdening consumers of a service with handling its failures.

Failures should not propagate to more components within a microservice as well as to other services and the entire system should stay responsive in the face of individual service failures. Components should be isolated from each other with each component containing its own failures thereby ensuring that parts of the system can fail and recover without impairing the system as a whole.

THE PRINCIPLES 12/13

- **Highly observable**

Maintaining a healthy production environment where a fleet of microservices forms an application's ecosystem, requires that the component services are highly observable. The environment must be discoverable. It should provide enough information to ascertain the state of the system.

"Monitor. In fact, monitor everything."

THE PRINCIPLES 13/13

- **Appropriate Security**

A service should implement the appropriate level of security based on the domain it encapsulates. The implication of this principle is that different services may implement different security mechanisms as the level of security applied is dependent on the functionality the service exposes.

SUMMARY

- Principles are an integral part in realizing the efficacy of microservices architecture.
- They reflect a level of consensus across an organization, and embody the spirit and thinking of the architecture while acting as guidelines to assist with decision making. They ought to be high level and not written in stone.
- While principles address the '*why*', standards mandate what ought to be done. There is however a natural relationship between principles and standards that takes the form of *standards* arising from *principles*.

REFERENCES

- https://enterprisersproject.com/article/2017/8/how-explain-microservices-plain-english
- https://www.computerworld.com/article/3427107/what-are-microservices-.html
- https://microservices.io
- https://www.ibm.com/cloud/learn/microservices
- https://medium.com/@k.wahome/maximizing-microservices-architecture-part-2-principles-5b8004b0ed09