Microservices
Unit 2
Chapter 3
Adopting Microservices in Practice

Q1)Discuss issues you may encounter when working at this macro-level view of the system.

(Solution Architecture Guidance).

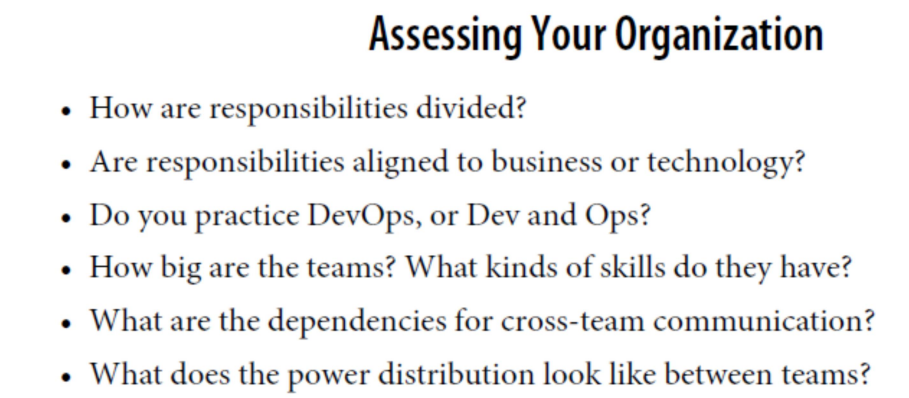
- **1.How many bug fixes/features should be included in a single release?**
- Since releases are expected to happen frequently, each release will likely be small.
- You probably can't box up 50 changes to a single service component in a week.
- We hear most organizations have a practice of limiting the number of changes.
- Netflix, for example, tells teams to make only one significant change per release.
- The biggest reason for limiting the number of changes in a release is to reduce uncertainty.
- If you release a component that contains multiple changes, the uncertainty is increased by the number of interactions that occur between those changes.
- The mathematical discipline of graph theory provides a simple formula to calculate those interactions: n(n–1)/2.
- Based on this, if you release a component that contains 5 changes and it causes problems in production, you know that there are 10 possible ways in which these 5 changes could interact to cause a problem.
- But if you release a component with 15 changes there is a potential for over 100 different ways in which those changes can interact to cause problems.
- Limit the number of changes in each release to increase the safety of each release.
- **2. When do I know our microservice transformation is done?**
- Technically, creating and maintaining a vital information system is never "done."
- One of the advantages of microservices is that change over time is not as costly or dangerous as it might be in tightly coupled large-scope release models.
- Trying to perfect "the system" is an impossible task since it will always be a moving target.
- Often arriving at some "final state" marks the start of accumulating "technical debt"—that status where the system is outdated and difficult to change.
- It helps to remember that everything you build today will likely be obsolete within a few years anyway.
- Since doing things in the microservices way means lots of small releases over time, you'll always be changing/improving something.
- This means you get lots of "done" moments along the way and, in keeping with the theme of microservices, are able to effect change over time "at scale."

Q2) Explain "Good service design is a byproduct of good organizational design."

(Organizational Guidance)

- **How do I know if my organization is ready for microservices?**
- You can start by assessing your organization's structure and associated culture.
- "Conway's law,": the design of a software system will mimic the communication structure of the organization that produced it.
- It is crucial to look at the organizational structure.
- How are responsibilities divided between teams?
- Are they aligned to business domains, or technology skillsets?

- At what level of the organization are development and operations divided?
- How big are the teams? What skills do they have?
- How dynamic is the communication and interaction between the teams who need to be involved in the delivery lifecycle?
- In addition to these organizational variables, you should evaluate the culture.
- How is power distributed between the teams?
- Is it centralized at a high level, or decentralized among the delivery teams?
- Answering these questions will help you understand what impacts these organizational factors will have on your adoption efforts and resulting successes.



- The ideal organization for microservices has small, empowered teams responsible for services that align with specific business domains.
- These teams feature all of the roles necessary to deliver these services, such as product owners, architects, developers, quality engineers, and operational engineers.
- The teams also need the right skills, such as API design and development, and knowledge of distributed applications.
- Organizations that mismatch any of these characteristics will pay a toll when attempting to apply microservice architecture.
- Teams that are not empowered will experience delays waiting for decisions to be made above their heads.
- Lack of business alignment will lead to cross-team dependencies, causing further delays and architectural deviation.
- Teams that are too large create incomprehensible code bases that impede and delay future changes.
- Lastly, if the team doesn't have the right skills to build API-fronted services using distributed concepts, costs could go up to cover training and/or contract hiring, or the solution could be dragged away from the microservices approach as existing resources retreat to their technological comfort zones.

Q3) Explain "Organization's culture shapes all of the atomic decisions that people within the system will make." (Culture Guidance)

- **How do I introduce change?**
- The challenge is to find small changes that can unfold in a way that creates large effects...
- If we make a mistake when refactoring our software we can always undo our changes, but when we make a mistake when redesigning the reporting structure in an organization the damage is not so easily undone.
- In order to apply a refactoring strategy to the organizational design, you'll need to:
- 1. Devise a way to test changes
- 2. Identify problem areas in your organizational design
- 3. Identify safe transformations (changes that don't change existing behavior)

- Your goal should be to do the same things, but improve the design of your organization so you can do them better.
- The goal in this step is to identify how software changes are introduced to the system, who implements those changes, and the type of coordination that is required for those changes to take place.
- For the microservice system we are especially interested in identifying opportunities to improve the efficiency of change.
- Gaining a total understanding of how your organization works may be too large of an initial investment to undertake, so in practice you may need to focus only on the changes that occur the most often for the components that are the most volatile.
- You should be looking for the bottlenecks that cause change to be expensive.
- Which processes result in a queue or backlog? Are there particular centralized functions such as audits, code reviews, and gating procedures that cause teams to have to wait? Are there any parts of your process flow that make it difficult for multiple changes to be introduced at the same time due to resource availability or a need for serialized process execution?
- Finding these bottlenecks will help you identify good candidates for process and organizational refactoring as they should yield a large benefit to the changeability and speed of release for the system.
-
- **Can I do microservices in a project-centric culture?**
- "products not projects" one of the primary characteristics for a microservice application.
- Typical project-centric cultures operate differently.
- Teams are formed to address a particular problem (e.g., create a new component, add a feature, etc.) and disbanded when that problem is solved. Often a good deal of knowledge about both the problem and the solution gets lost when the team disbands.
- And, if there is a need to readdress the same problem, or make additional changes to the same component, it may be difficult to re-create the team or recover the lost knowledge.
- In truth, it is quite difficult to adopt the microservice style if you need to operate in this type of culture.
-
- **Can I do microservices with outsourced workers?**
- With the right outsourcing structure, a microservice system may lend itself well to being developed by an external organization.
- By embracing a decentralized way of working and standardizing on the output and processes of service teams (containers and APIs), the outsourced development team can be given enough autonomy to build a service that meets the capability requirements of the owning organization.
- The teams should be the right size, built to last for the life of the service, and composed of workers who are skilled, experienced, and capable enough to make good design and implementation decisions autonomously.
- Culture becomes an important element in deciding which companies or people should be chosen to support the outsourcing model.
- Teams should be dedicated to services, workers should be capable of working autonomously, and speed of high-quality delivery should be the primary metric for success.

Q4)Explain the Tools and Process Guidance.

- **What kinds of tools and technology are required for microservices?**
- The ideal technological environment for microservices features cloud infrastructure, which facilitates rapid provisioning and automated deployment.
- The use of containers is particularly useful to enable portability and heterogeneity.
- Middleware for data storage, integration, security, and operations should be web API-friendly in order to facilitate automation and discovery.
- The ideal programming languages for microservices are API friendly as well, and should be functional while also matching the skillsets of your organization.
- It is particularly useful to provide tools for developers that simplify their tasks yet incorporate constraints that encourage good operational behavior of their resulting code.
- Lack of cloud infrastructure will lead to deployment delays and inflexible scaling.
- Lack of containers— or reliance on older virtualization or app servers—could increase the cost of resource utilization and lead to quality issues resulting from inconsistencies across environments.
- Middleware that assumes strict centralized control will break the decentralized organizational model.
- Lack of developer tooling consistency could lead to duplicate work and lack of visibility or resiliency in the overall system.
- Finally, a large dependency on legacy applications could limit the ability to make changes.
-
- **What kinds of practices and processes will I need to support microservices?**
- The ideal software development lifecycle for microservices is based on a product mentality using Agile principles, which includes continuous integration and continuous delivery and features a high degree of automation in testing, deployment, and operations.
- Project-focused delivery assumes static requirements and heavyweight change control, both impediments to fast software delivery.
- If change frequency is increased in an environment that has a legacy of change intolerance, many of those overweight processes can stick around, slowing down delivery, and introducing procedural fatigue as a new risk.
- Lack of automation in the deployment lifecycle will have a negative compound effect on speed to market, and lack of automation in operations will make it harder to deal with the operational complexity of a distributed environment.
-
- **How do I govern a microservice system?**
- There are typically three ways in which you can address security and governance requirements in a microservice system: centralized, contextual, and decentralized.
- **1. Centralized controls:**
- If we have a need to authenticate, authorize, and audit messages before they are processed, the most common architecture pattern is to implement some form of central security enforcement component within the architecture.
- Unfortunately, services like access control are likely to be used by every service in the infrastructure, which results in a common component that all of other services will grow dependent on. In other words, a bottleneck can develop.
- A centralized security component risks putting our system into a state of mechanical organization or centralized control.
- **2. Decentralized controls:**
- The implication here is that the individual microservice teams will need to manage an infrastructure that includes security mechanisms that are bounded to the service itself.

- The organization may standardize on the particular components and libraries that are to be used in every microservice, but it will be the teams themselves that are responsible for implementing security components and con- figuring them accordingly. It naturally follows that someone on the team must also take on the role of becoming the security expert for the service.
- **3. Contextual controls:**
- A third approach that an organization can take is to define subsystems within the microservice architecture.
- Each subsystem may contain multiple services and their services within the subsystem are able to share common resources such as access control.
- Again, the organization may mandate the nature and requirement for these security components to be in place, but it is up to a subsystem service team to own and manage the configuration for the security component.
- While decisions about how and what to secure will be dependent on the risk profile of your organization and nature of the application you are building, the decision about who will manage the security implementation and where it will be implemented will have a big impact on your ability to optimize for the system behavior that you want.

Q5) Explain Services Guidance.

- **Should all microservices be coded in the same programming language?**
- The short answer is "no." The internal language of the component is not as important as the external interface—the API—of that component.
- At the same time, many companies we talked to constrained the number of languages supported in the organization in order to simplify support and training.
- While a polyglot environment has advantages, too many languages results in added nonessential complexity system developers and maintainers need to deal with.
- **What do I do about orphaned components?**
- Over the life of a microservice implementation teams will come and go, and sometimes a team might disband and this can result in an "orphaned" microservice.
- It's not a good idea to just let a service run along without someone to care for it.
- When a team is about to disband, that team needs to designate a new "owner" of the microservice component.
- This might be one of the existing team members It might be some other team that is willing to take care of it.
- Or it might be someone who has taken on the special role of caring for "orphaned" services.
- But someone needs to be designated as the "owner."
- It's not safe to allow orphaned services to run in your infrastructure.