Benefits:

* Small services allow you to adjust them, and rapidly write, deploy, reuse, and replace them.
  + Faster delivery
  + Small services that do one thing well
  + Can be independently deployed
* Currently our services are very large which makes it difficult to identify and quickly fix issues.
  + You may have to re-deploy an entire application for a single change to matter how small
  + Leads to infrequent large risky deployments with a lot of overhead
* User doesn’t need to get familiar with whatever different system is providing the service, and if and when it is replaced, they won’t notice
* By building in a service layer it allows interchangeability of the system providing the service, allowing the enterprise to easily plug in new applications when available and transition off the legacy product
  + Lower risk and cost in upgrading and replacing
* By building a service catalog it should provide us the potential cost savings to eliminate duplicative products
* Reduced risk of change & introducing bugs given isolated micro services
* Lower risk to introduce new tech since it only impacts a small service
  + Different teams can use different technology stack for the microservices they manage
* Lower risk and cost of replace systems and migrating and operating in the cloud
  + More flexibility in cloud deployment, some services to the cloud, some in-house, depending on pricing, security, tech, and other factors
* Smaller, more productive teams to do focusing on smaller services

Pitfalls

* Many organizations find microservices too complex, expensive and disruptive.
  + Often microservices complexity is better complexity than that of a giant monolithic software application
* Need the experience in deploying and managing distributed applications and ideally microservices
* Extra overhead to do comprehensive analyze to determine return on investment
* Integration and end-to-end testing can be challenging with more pieces
* Determining how to break of the services can be challenging
* Can be overkill starting with many applications, often starting with a monolith is the right way to begin

Approach: Start small and Scale

* Take it slowly. Initially use microservices only where you need to — for functionality with agility or scalability requirements. Direct teams to experiment with the new patterns and technologies and build expertise. Direct a team to start building a sandbox.
* Choose a low risk project to start with.
* Create version 1 as a monolith, then slice off microservices as they make sense. This also gives time to build up the tooling and infrastructure incrementally.
* Lift and shift an application into a container, and deploy that to a container management system. This means taking an existing app and moving it into a container as a whole, without trying to break off microservices yet. This will allow you to get it into cloud and other microservices infrastructure, and make it easier to break up from there.
* Expect to implement a microservices infrastructure to provide the outer architecture capabilities needed to build, deploy and operate fleets of microservices. PaaS (Platform as a Service) and container management platforms can provide a core, but you’ll need to supplement this with capabilities such as API gateways, service discovery, monitoring and telemetry, build and test automation, messaging and data persistence. A serverless approach can be a quick way to get started.
* Bring in outside expertise in distributed applications, container, cloud and microservices if this is possible. If not, plan for learning time.

1. Build a list of service catalog
   1. Limit initial scope to incident facilitation