Building Microservices Ch3

Splitting the Monolith

① Have a goal try the simple stuff first, track your progress against that goal

② Incremental Migration start small, get it into production + learn from your mistakes

③ The Monolith is rarely the enemy focus on the benefits from the new architecture. "Real system architecture is a constantly evolving thing that must adapt as needs + knowledge change."
* in rare circumstances the monolith needs to be destroyed.
- dead/dying technology, tied to infrastructure that needs to be retired, or an $$ 3rd party system to get away from.

④ Dangers of Premature Decomposition, if you don't have a clear understanding of the domain you're in danger

⑤ What to split first?
- Depends on goal (ex. Scale, speed to release)
- Can use tools like Code Scene to identify ~~tools~~ frequently changing code
- balance how easy the extraction is vs. the benefit of extraction

⑥ Decomposition by Layer
- Make sure UI decomposition doesn't lag too far behind the backend
- Can look at all layers of your slice (UI, backend, data)

⑦ Code First Extraction
- if you do code first, be sure to look at if you can eventually extract the data too. It's a problem if you can't.

⑧ Data First Extraction
- Can be useful if you're not sure the data can be extracted
- ↓ risk, can deal upfront w/loss of enforced data integrity or lack of transactional operations across both sets of data

⑨ Useful Decompositional Patterns
a) Strangler Fig Pattern (Martin Fowler), intercept calls + direct to new service expand coverage over time.
b) Parallel Run, run monolith + μservice side by side + compare results
c) Feature Toggle, to switch b/w monolith + μservice

(10) Data Decomposition Concerns

a) Performance, joining in μservice tier is just not going to be as fast you can bulk grab data or use aggressive caching

b) Transactions, split data loses ACID Transaction guarantees, there are other ways to manage state changes, but they're complex

c) Tooling, need to engage w/tools like Flyway or Liquibase to manage refactoring or changing the schema

d) Reporting Database, have a central location the other databases are fed to.

i) we still want information hiding, so the reporting db may not have <u>all</u> of the info from all of the databases

ii) the reporting db should be treated like any other μservice & will need to be maintained & external contract retained even after changes