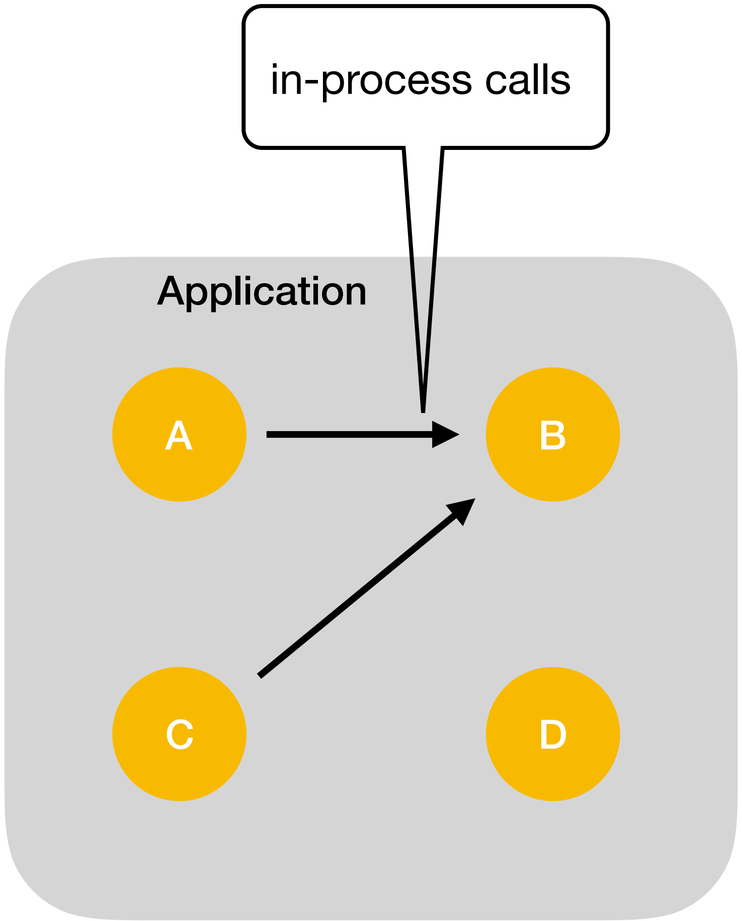
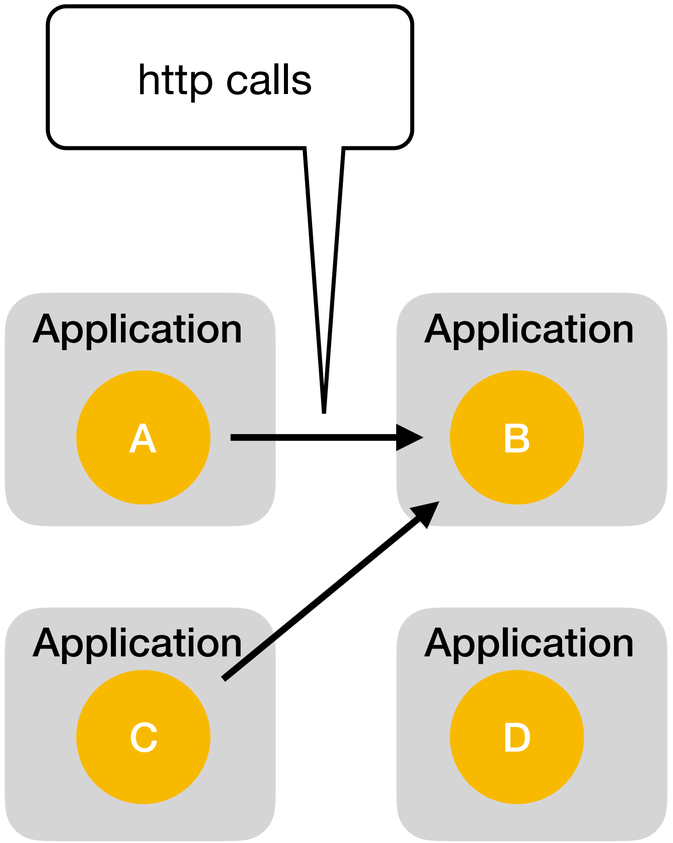
Effects on Testing

- With a monolith, components communicate directly via method invocations and interfaces definitions
- Contract is not always clear, or enforced
- Correctness of implementation verified with unit tests



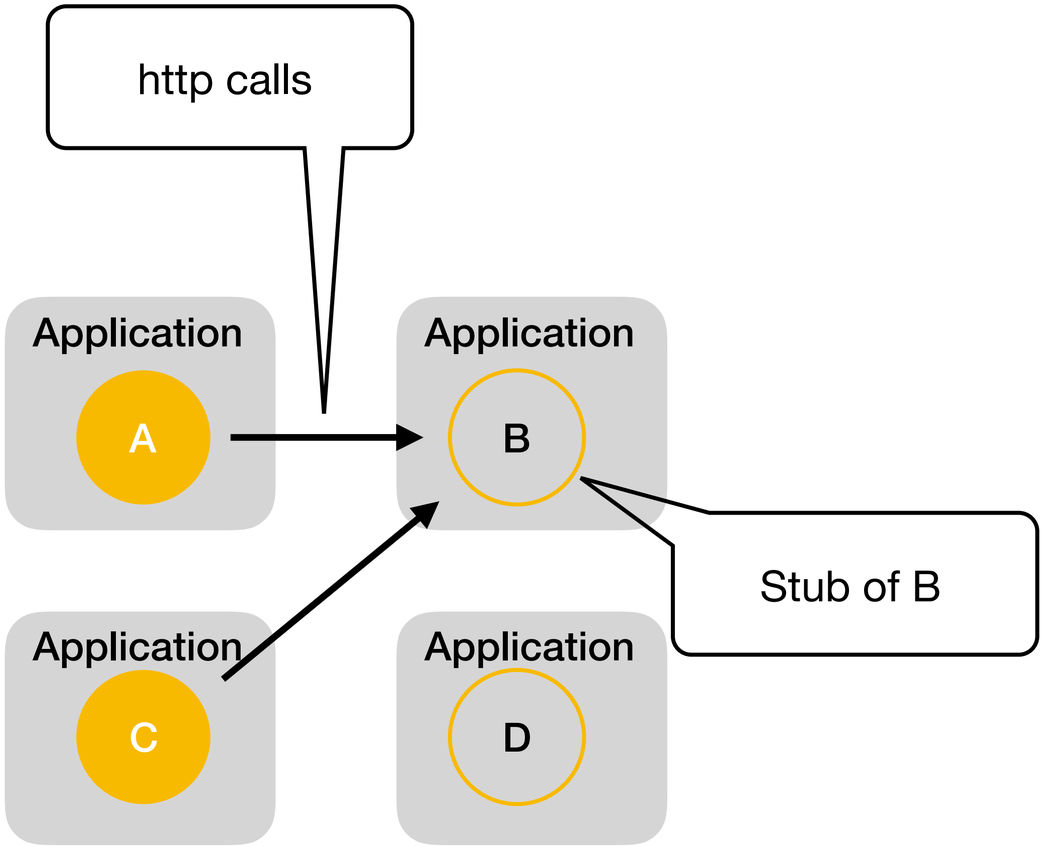
- With microservices, service interactions become integration points
- The contract between components can be enforced by integration tests



Testing

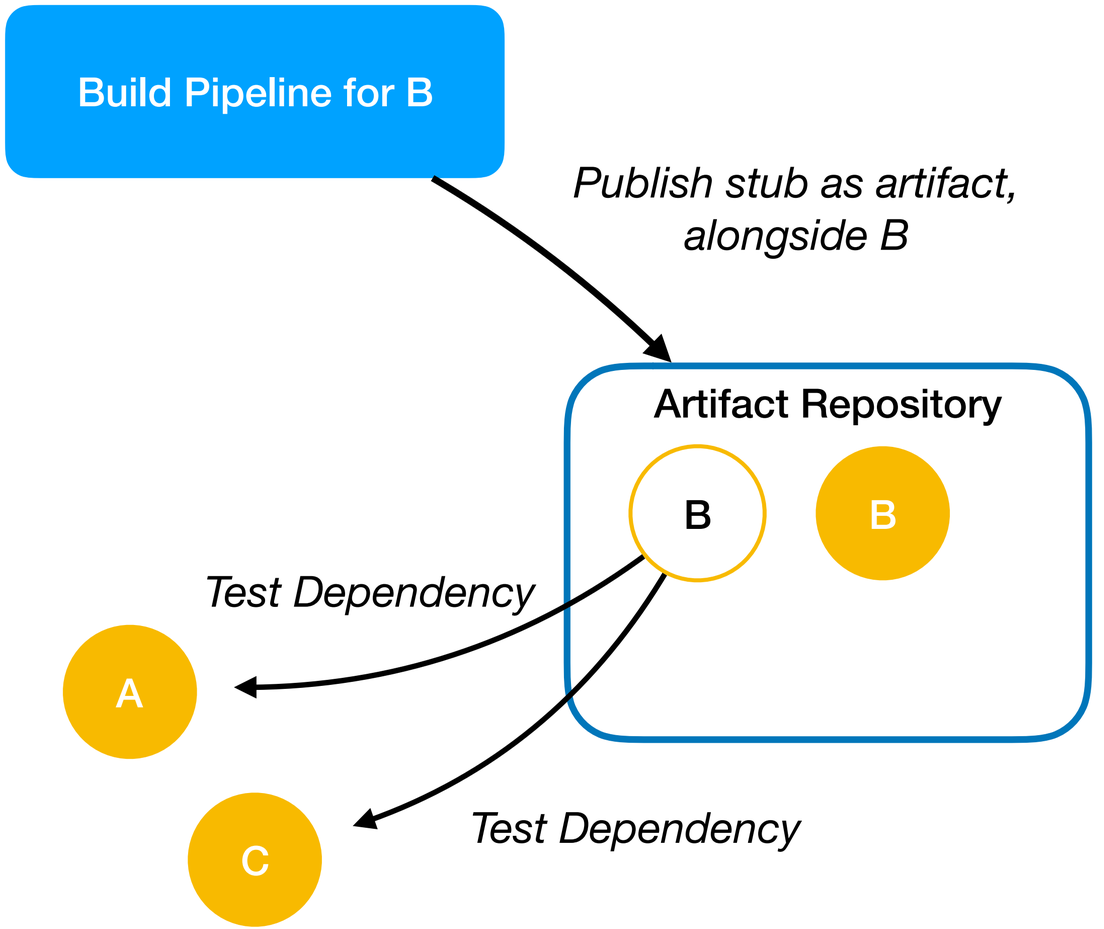
- Problem: tedious to stand up all collaborating microservices just to test these interactions

- Solution: define and run stubs that mimics the behavior of the service in question



Responsibilities

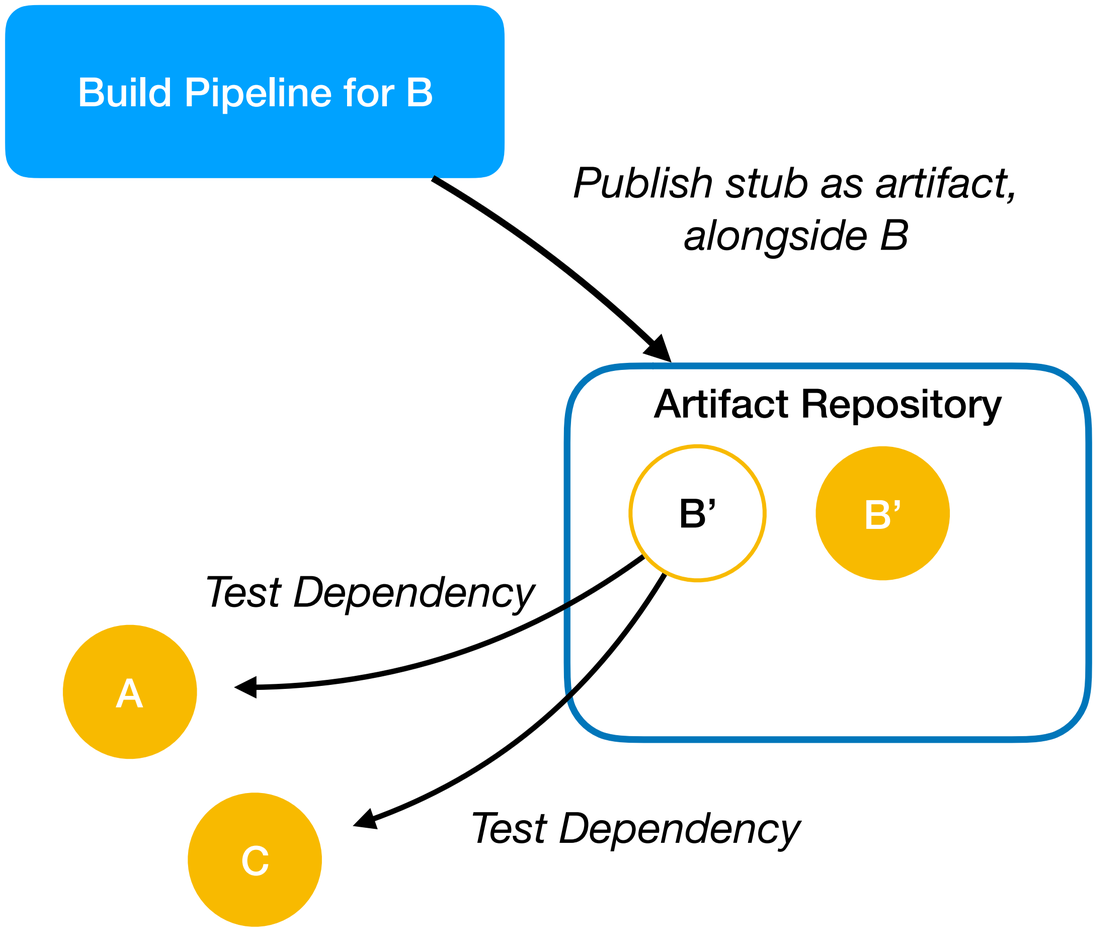
- Problem: multiple teams each develop their own stub implementations for a backing service. Inefficient, relationship is inverse of what it should be
- Solution: team building a microservice should publish stubs for all clients to use



Contract Changes

- Problem: service contract changes can break clients who adhere to previous version of contract
- Solution: keep implementation of stub in sync with implementation of real service.

Every time a change is made to the contract for a service, that change must also be reflected in the stub



Consumer-Driven Contracts

- Problem: service implementor often does not take into account, or is unaware of how their service is used by multiple, different clients

- Solution: have clients write and contribute their tests against said service to the team that maintains the service. Clients' tests become part of the service's test suite

This ensures that each time a change is made to the implementation of the service, clients have a say in ensuring that their use of the service is not broken by that change

Contract Evolution

- Clients can go beyond ensuring correctness, and drive the evolution of the contract

- Service clients are the ones who exercise an API: they have a specific use-case, a sense for how the current contract “fits”, and ways to improve and evolve it.