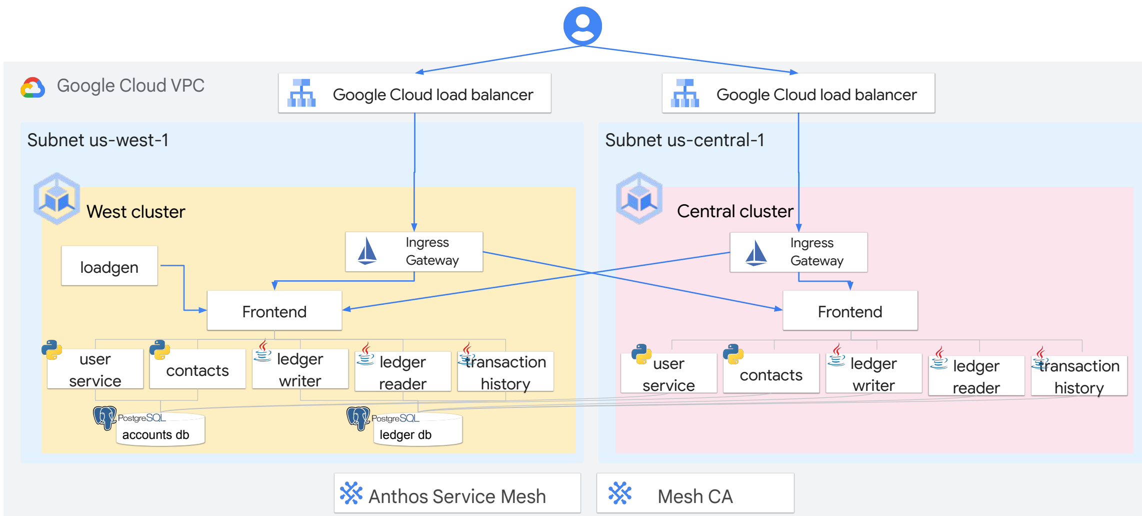
Today's agenda

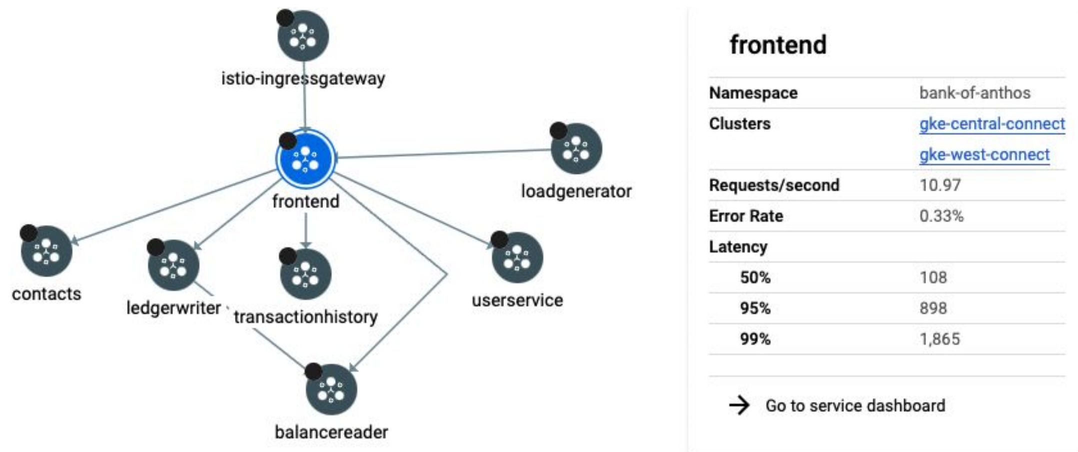
Lab review: Multi-cluster GKE with Anthos Service Mesh



Google Cloud

In this lab, we saw a multi-cluster GKE environment running Anthos Service Mesh. Services were deployed into multiple clusters as distributed services.

Lab review: Routing distributed services across clusters



Google Cloud

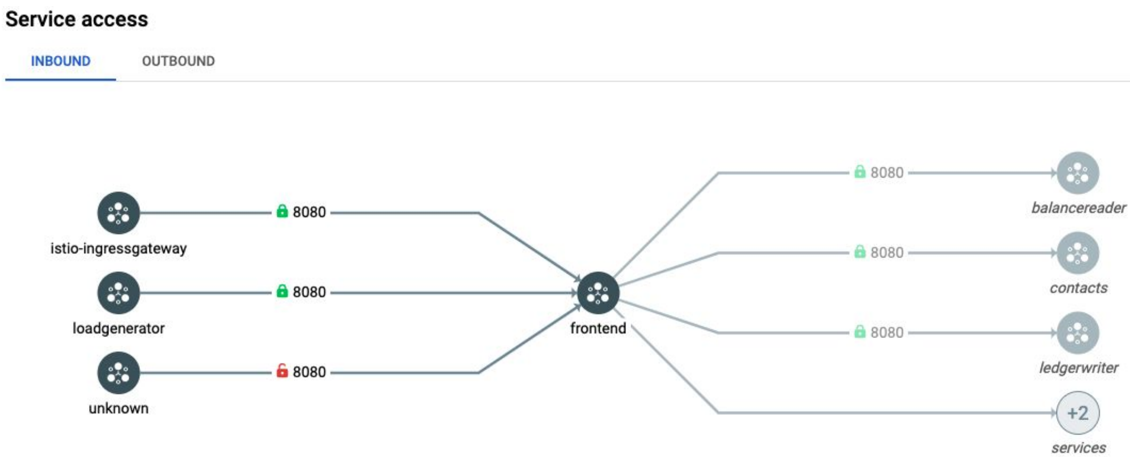
Distributed services provide multi-regional availability and remain up even if one or more GKE clusters are down, as long as the healthy clusters are able to serve the load.

Lab review: Service telemetry and observability

Google Cloud

All service telemetry was collected and aggregated in the Anthos Service Mesh dashboards, and made ready for consumption, so that you can analyze data over time and make sure your services meet your Service Level Objectives, or SLOs.

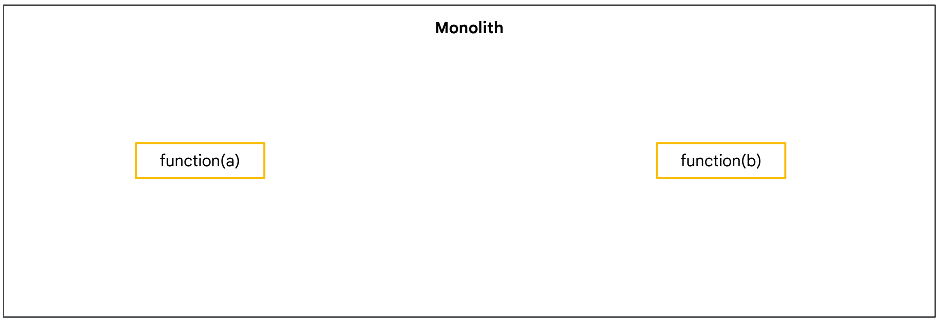
Lab review: Secure service-to-service communication



Google Cloud

All traffic between services was encrypted over mTLS and you had the possibility to implement and enforce policies to and from every service.

In monolith applications, functions call each other directly

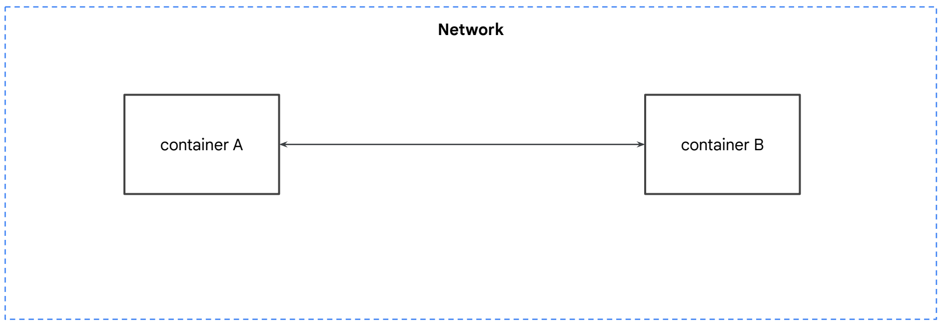


In monolith applications, functions call each other directly..

With microservices, direct calls are not possible

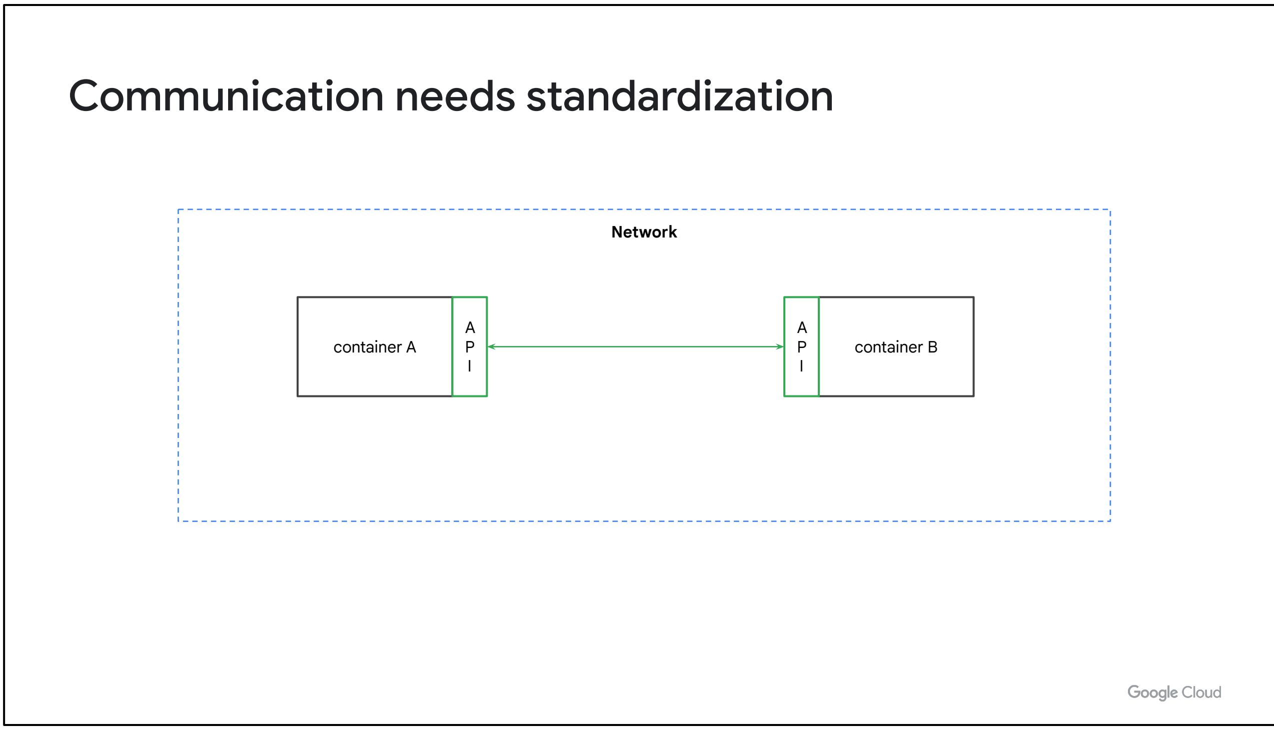
When moving to microservices, the functions may be encapsulated in separate services, and those services might be containerized. So, no direct calls can be made from function a to function b.

Communication happens over the network

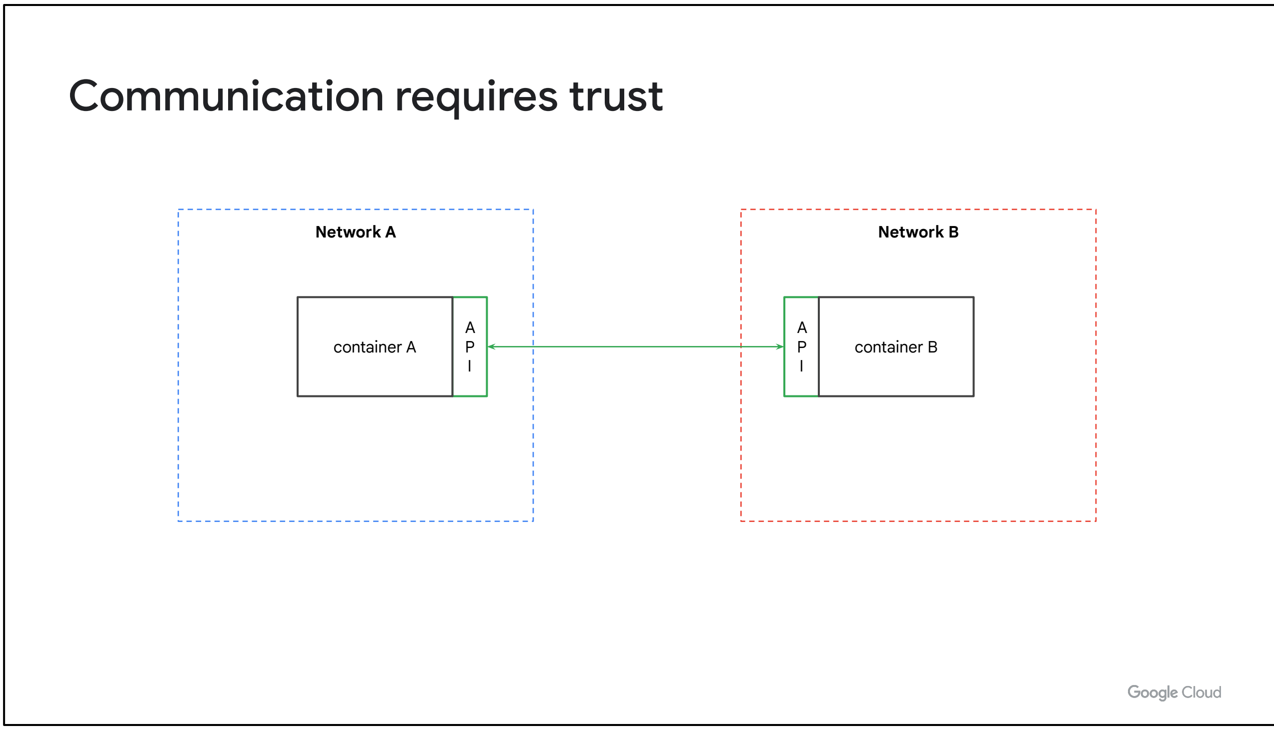


Google Cloud

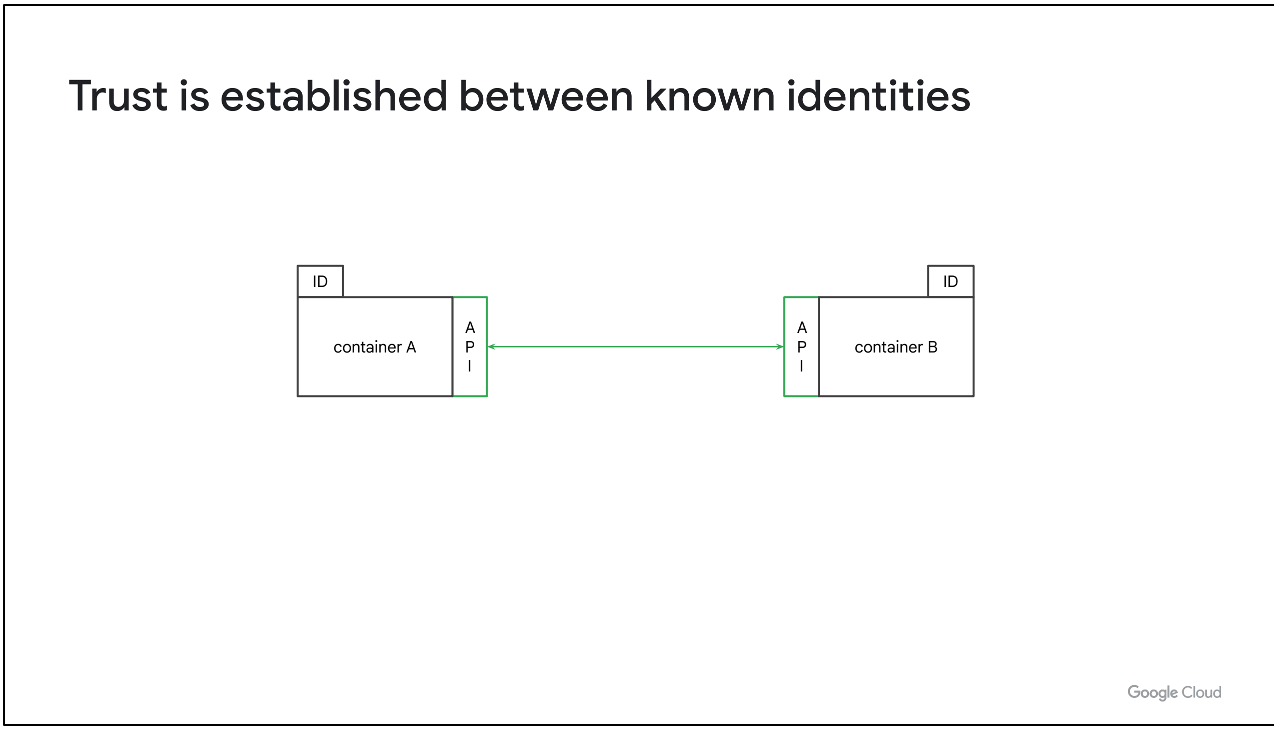
The services are no longer necessarily inside the same computer, yet they still must communicate with each other. This requires networking.



We want to expose the two services via a contract. We abstract away how we handle the communication and standardize the way that the services communicate.



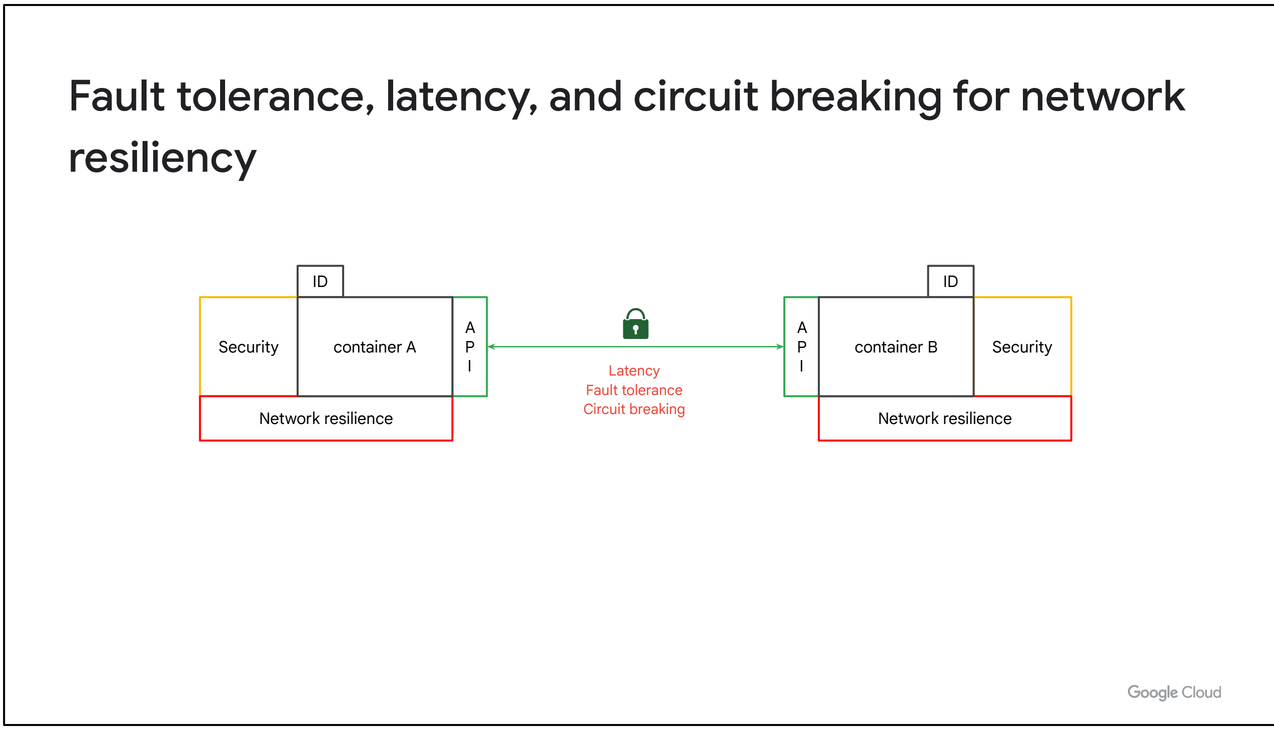
You don’t want to assume that just because the two containers are on the same network, that Container B should trust Container A. Rather, you want Container A to go through authentication and authorization whenever it calls the API exposed by Container B.



So, each container needs to be provided an identity.

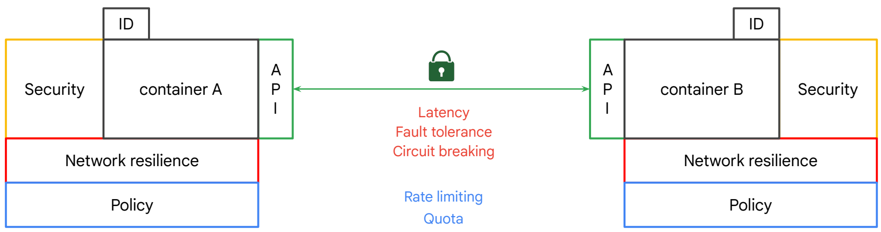
Security mechanisms to authenticate and authorize requests

And there needs to be some security scaffolding that can request/accept the identity of a calling service, and handle authorization.



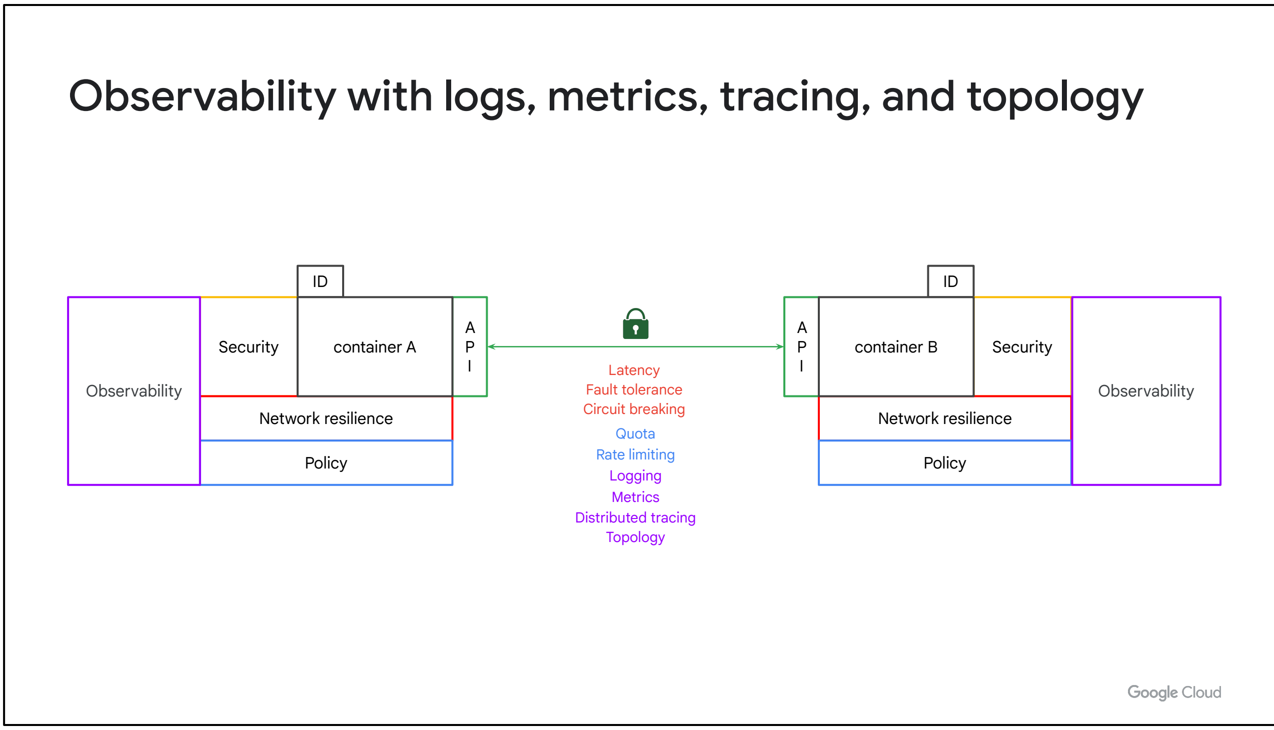
In addition, developers and architects must think about fault tolerance, latency, circuit breakers, etc. This requires additional scaffolding.

Policy enforcement and rate limiting

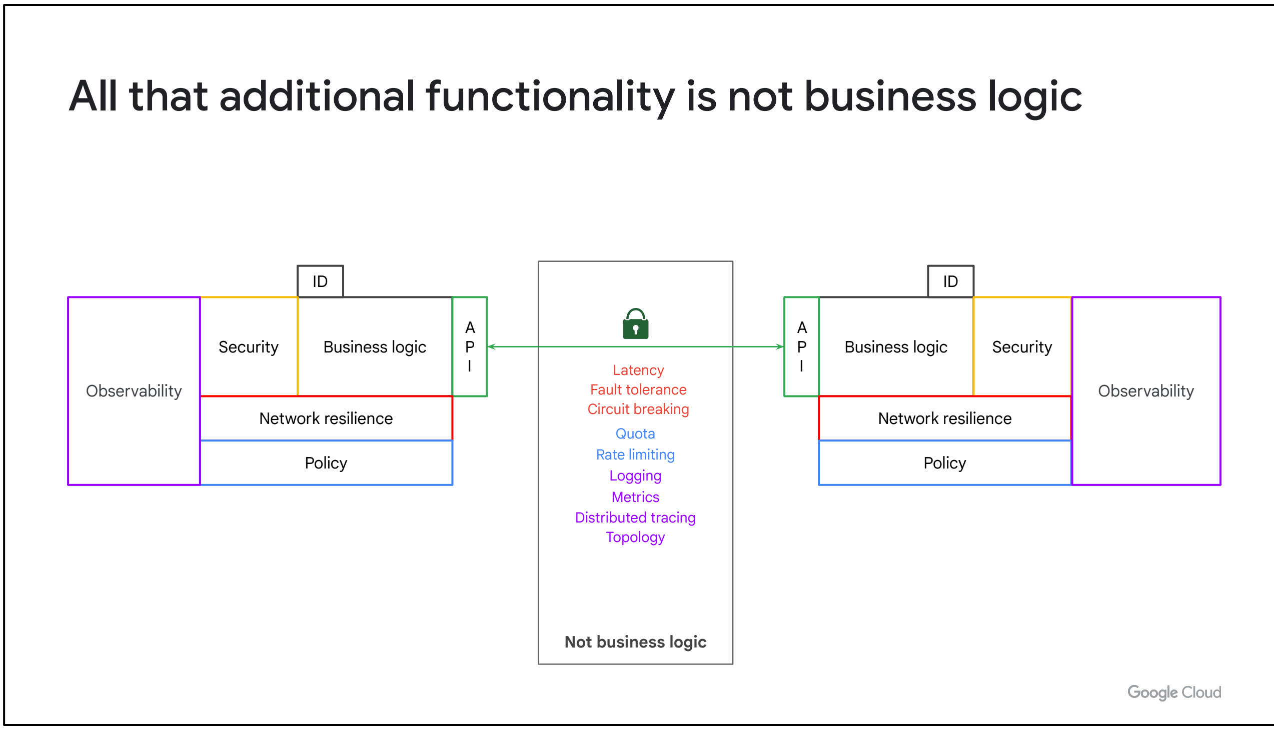


Google Cloud

The application may also need to enforce quotas and rate limiting and other API access policies. Thus, more supporting functionality needs to be in place.

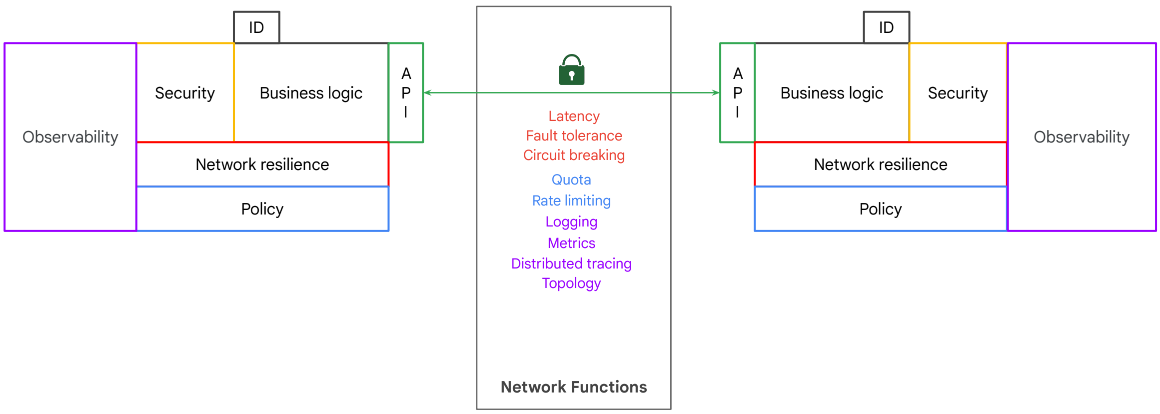


Lastly, if you want observability data like logs, metrics, tracing, and topology, you need yet more supporting functionality.



So, there is a lot of technical functionality that's required for successful operations of your application, and it's replicated for every instance of your service, but it really has nothing to do with the business logic of the services themselves.

Network functionality should not be handled at the application layer



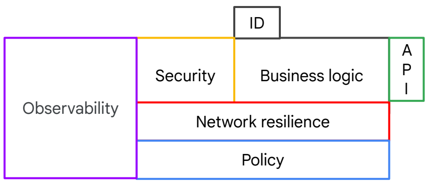
Google Cloud

This is really networking functionality, and perhaps can be handled at the network level rather than the application level.

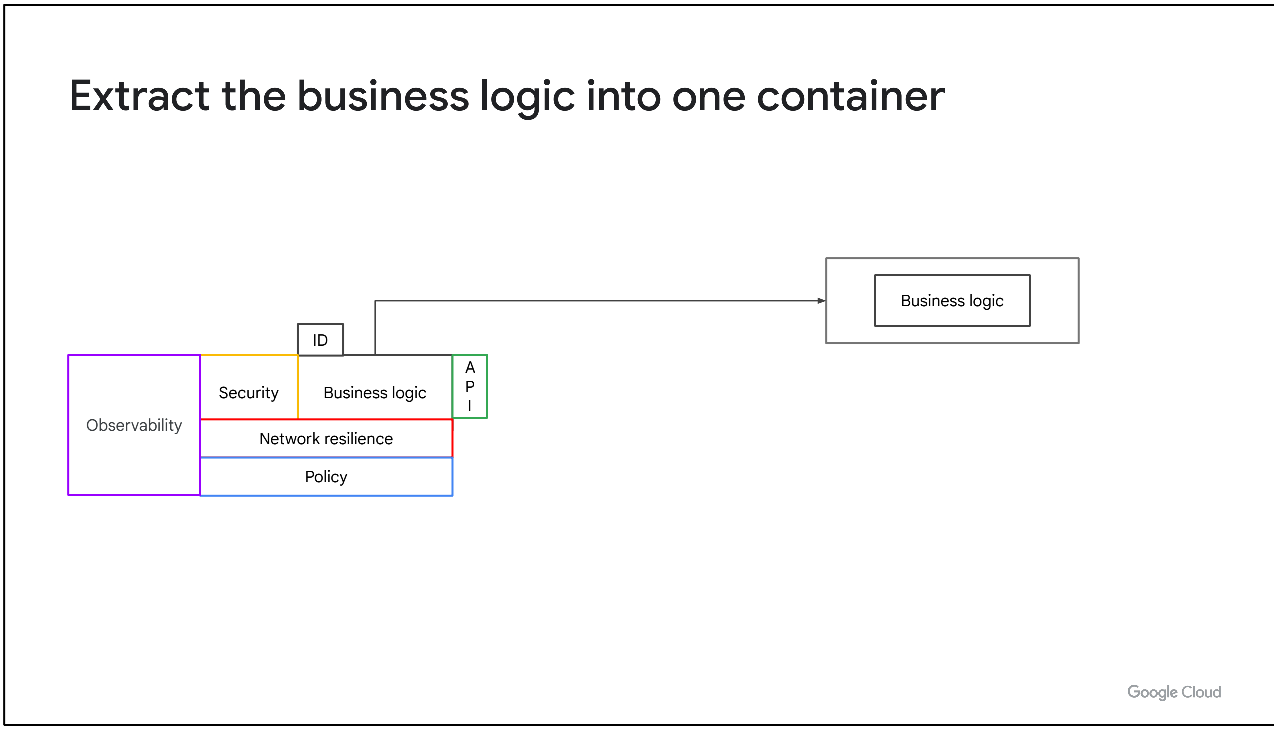
A service mesh separates applications from network functionality.

A service mesh separates applications from network functionality.

Take a replicate of a service

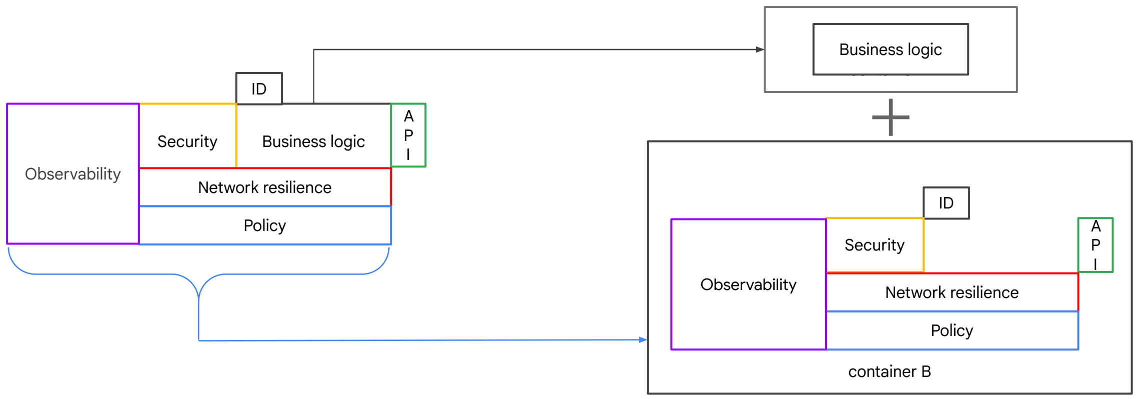


One approach is to take a replica of a service.



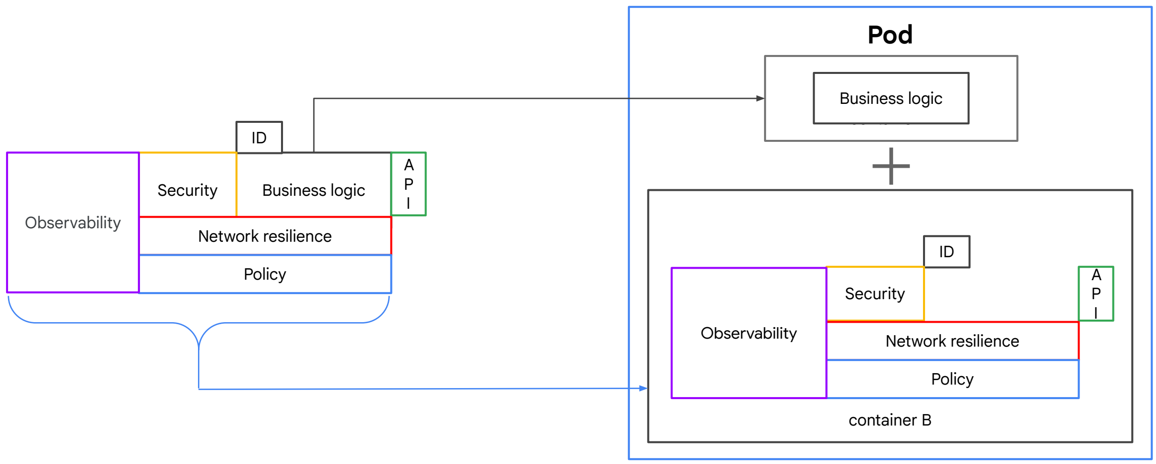
Extract the business logic into one container.

Extract all the network functionality into a separate container

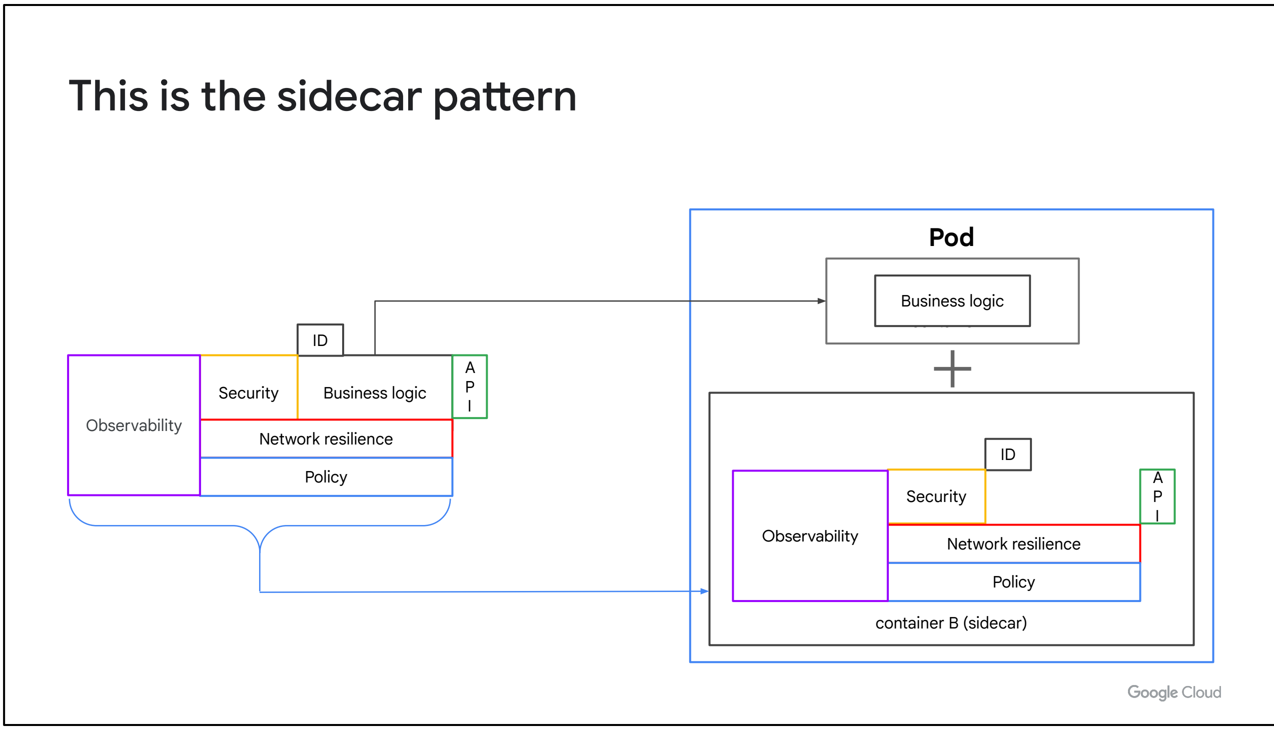


And then extract all the network functionality into a separate container.

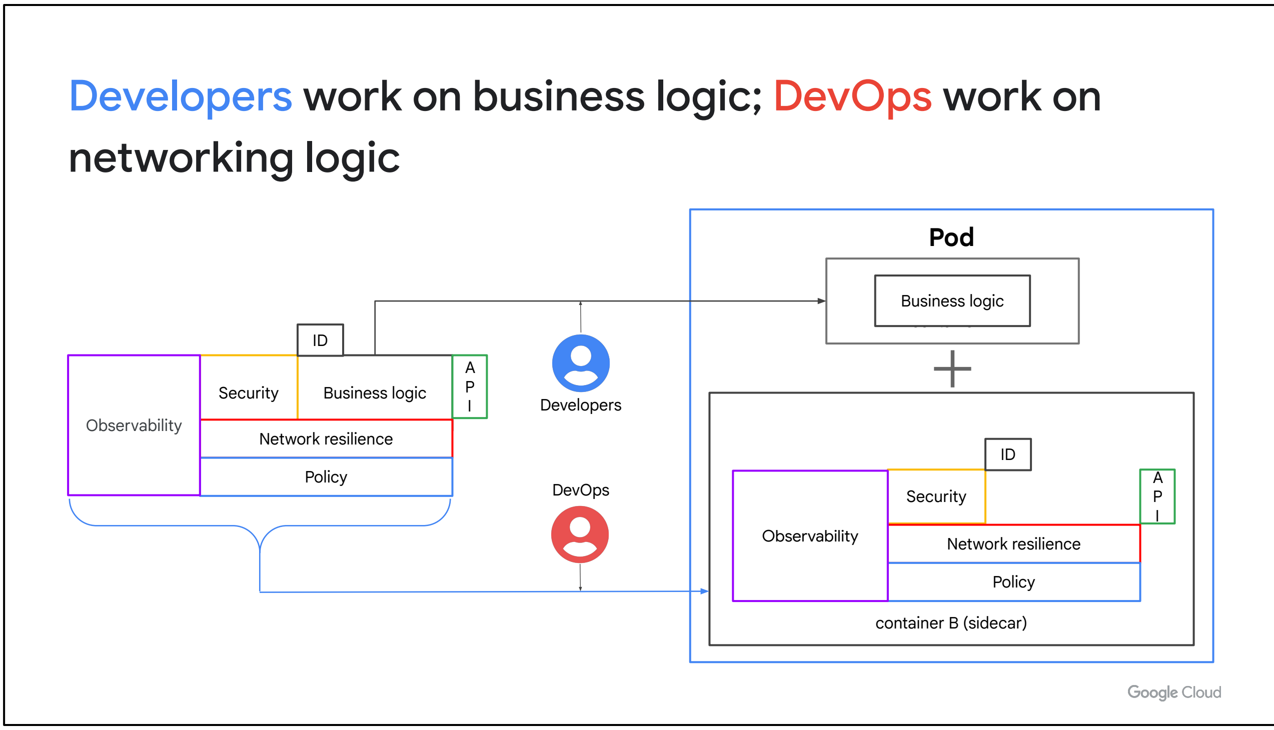
In Kubernetes, you can place two containers on a pod



In Kubernetes, you would place the two containers together into a pod, which represents one instance of your service.



This pattern of marrying a business logic container with a utility container that handles additional technical functionality is called the sidecar pattern.



This allows developers to focus on implementation of business logic, without having to invest in all the surrounding technology. Meanwhile, DevOps/SRE teams can focus on building all the technical scaffolding that adds valuable functionality around the business logic.