HashiCorp
Consul

Consul Service Mesh

Distributed service networking layer to connect, secure,

Speaker notes

Oh hey, these are some notes. They'll be hidden in your presentation, but you can see them if you open the speaker notes window (hit »S« on your keyboard).

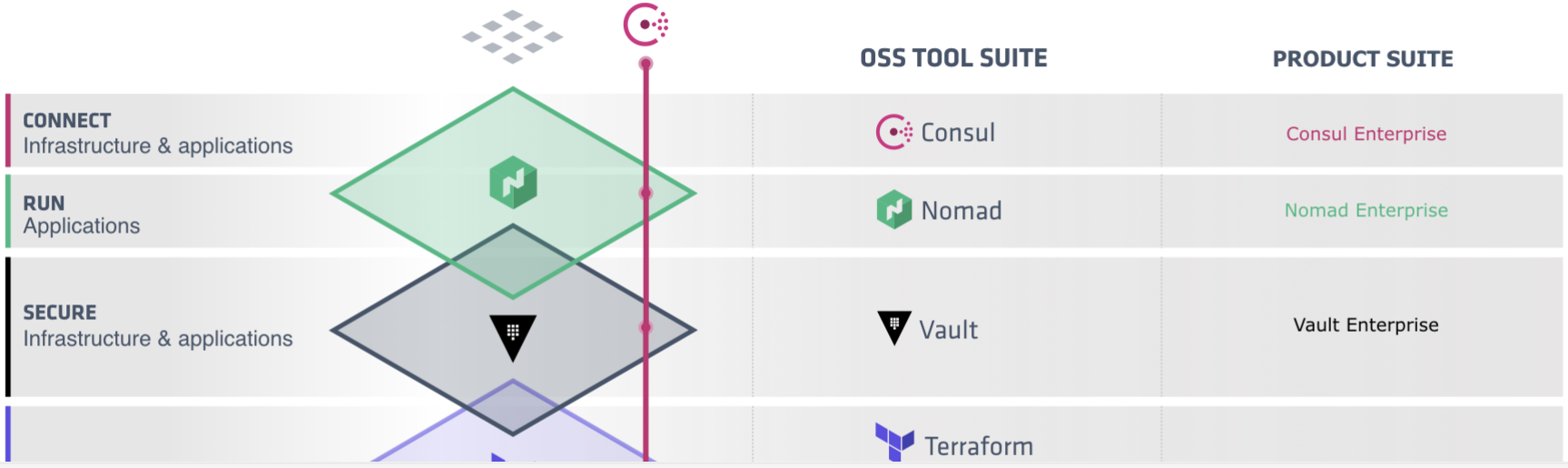
Presenter Name

Title, Affiliation
@socialmedia

Speaker notes

[Update slide with your photo, if desired, otherwise remove, your name, affiliation, and social media info. Update speaker notes with any biographical information you want to include.]

HashiCorp Suite



Speaker notes

HashiCorp's goal is to make tools that support application delivery. **Layers.** First, how do we provision the underlying infrastructure? Terraform, along with Packer and Vagrant. How do we secure this infrastructure? Vault. How do we run applications across multiple environments? Nomad. How do we connect and wire all these things together across multiple environments? Consul. HashiCorp Tools are designed to be Simple, Modular, Composable. (See Tao of HashiCorp.) They work great together, but also are designed to work with your existing infrastructure and tooling. Open source workflows address technical complexity Enterprise workflows address organizational complexity

From Monoliths to Microservices

A trend toward dynamic infrastructure

Speaker notes

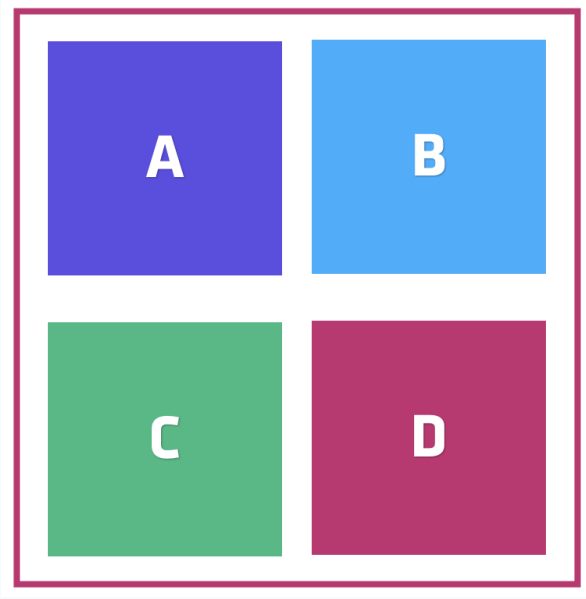
One thing we're all noticing, regardless of what stage in the journey we're currently at, is this trend away from monolithic, static infrastructure towards microservices and dynamic infrastructure. What do I mean by this? Let's take a look at where we're coming from...

Monoliths: How do they even work?

Speaker notes

A lot of us have experience deploying monolith apps. Let's take a look at how those often work.

Monolith

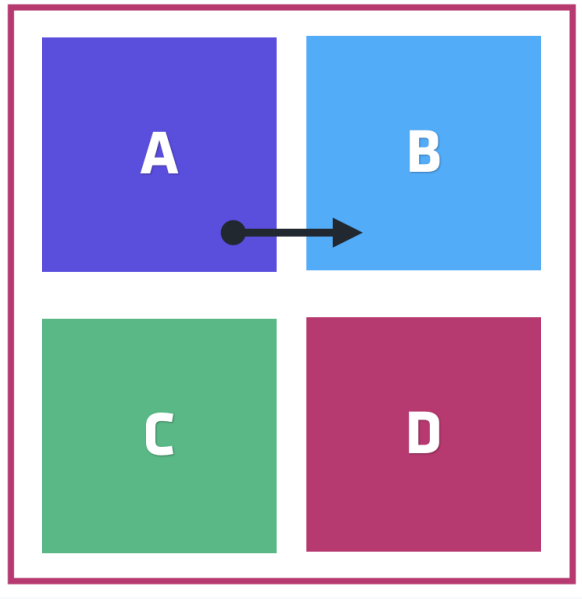


Many subsystems deployed as a single application.

Speaker notes

In a monolith, we have several many subsystems deployed as a single application. Possibly on a dedicated or virtualized server that was deployed and is maintained manually or with a collection of customized scripts.

Monolith

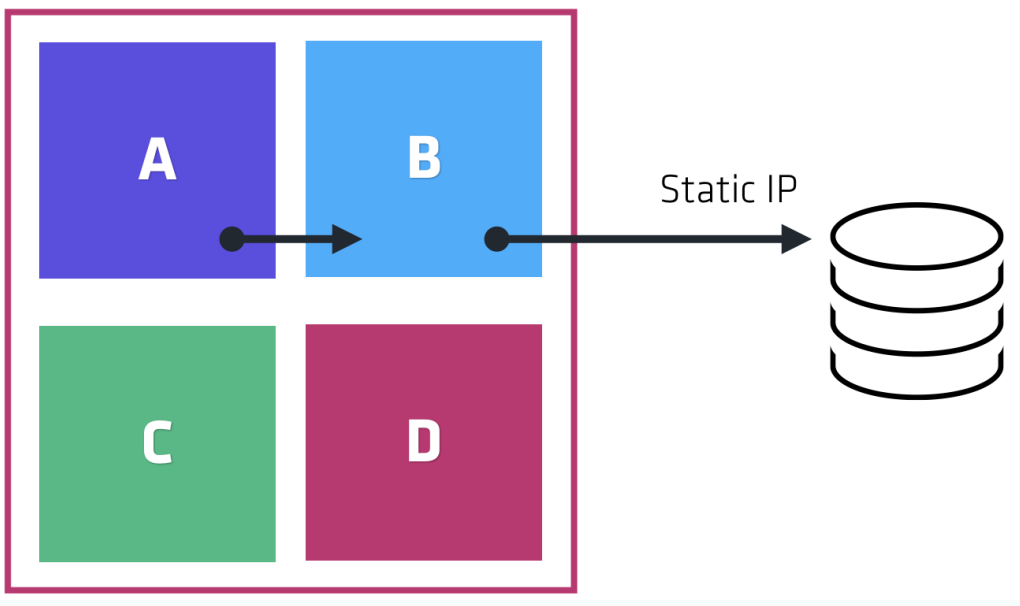


Subsystem calls stay within node. No network calls.

Speaker notes

When subsystem A wants to call subsystem B, this is done by an in-memory function call. Data is shared in process, and we don’t need to use the network.

Monolith

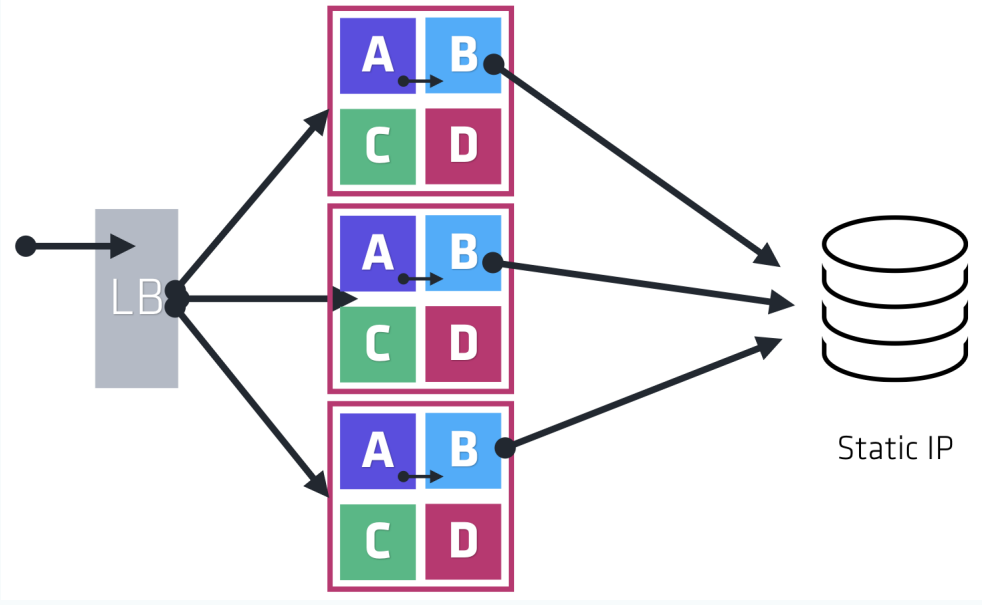


Static IP addressing typical for network calls.

Speaker notes

When a subsystem needed to communicate outside of its node, often static IP addresses were used.

Monolith

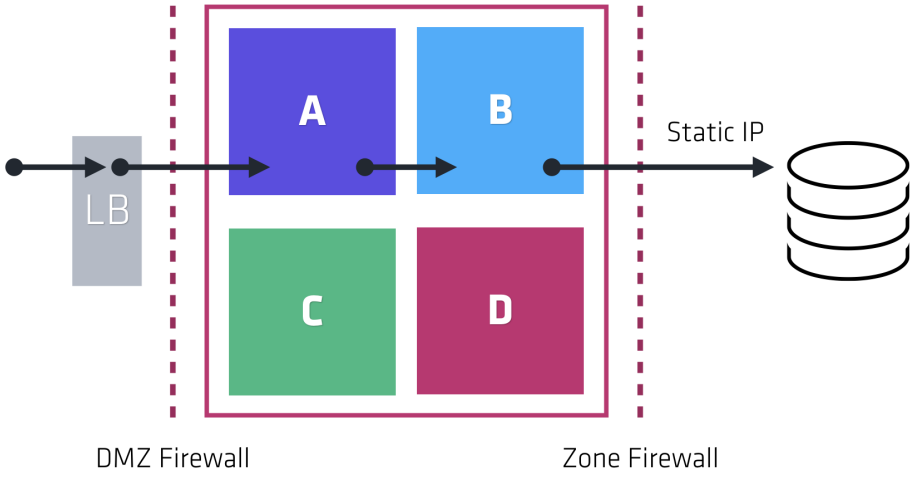


To scale, deploy many copies with load balancer.

Speaker notes

As we need to scale, we would deploy more copies of our monolith and use a load balancer to spread traffic.

Monolith



Firewalls for security.

Speaker notes

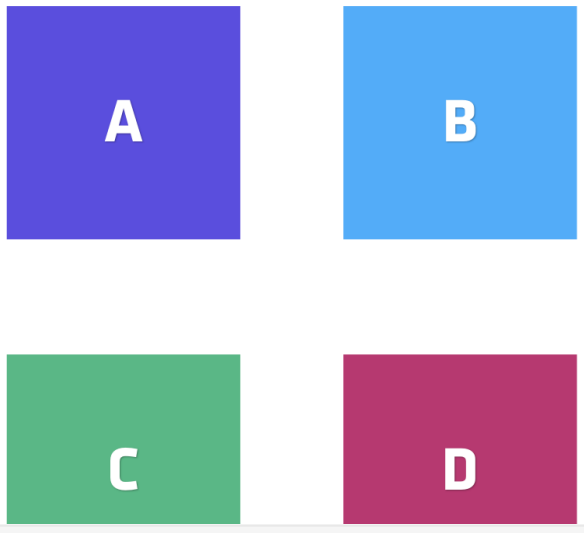
To secure our infrastructure, we use firewalls to ensure traffic only flows between the appropriate zones. A DMZ firewall to [something something] and a zone firewall to [something something].

What about Microservices?

Speaker notes

Okay, now how about microservices?

Microservices

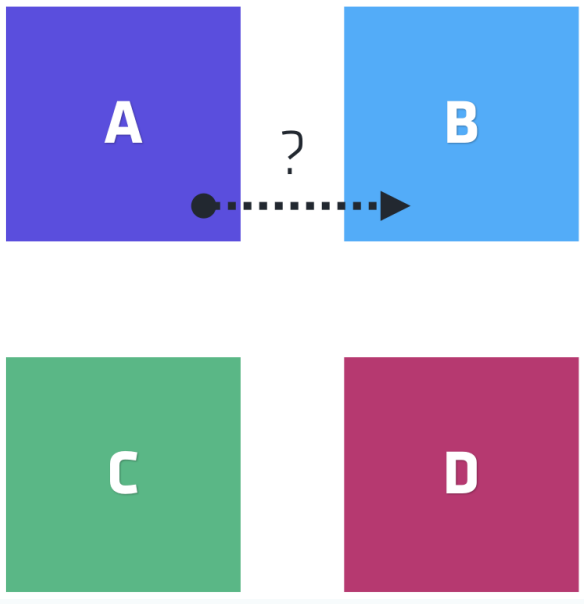


Subsystems now deployed separately. More agile.

Speaker notes

Now let's look at how microservices architecture [deployments?] works. With microservices, you're breaking out these subsystems that were previously deployed together, and deploying them separately, possibly on different machines, or even data centers. The sub systems now talk [interact with?] to each other via networked APIs. Decoupling systems into microservices provides greater operational efficiency. It allows us to get away from the "lowest common denominator" w/r/t development & deployment cycles. Each subsystem can move at its own pace.

Microservices

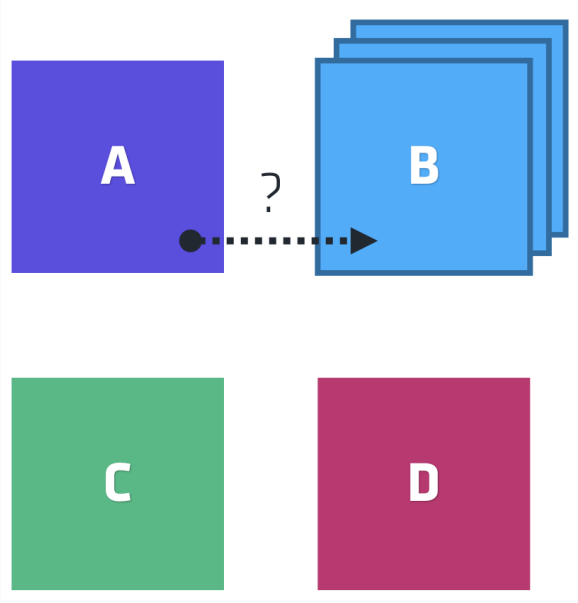


Agility comes with new operational challenges.

Speaker notes

While we gain agility during development, there is no free lunch! We've now adopted a set of new operational challenges. For example, in a monolithic architecture, when A wanted to call B, it was a simple in-memory function call. Now B must be reached over the network which means A needs to know how to reach B.

Microservices

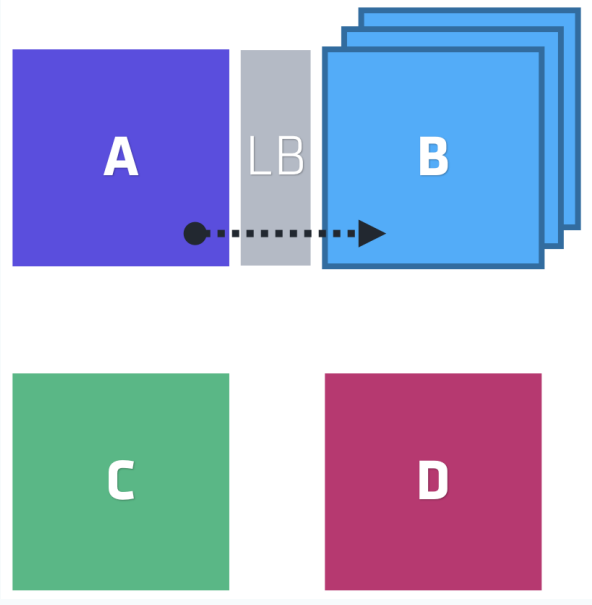


e.g., Static addressing difficult to scale with multiplicity of instances.

Speaker notes

We don’t want to use a static address for B for several reasons. We may want multiple copies of B for scaling or availability. We also can’t depend on static IPs in a cloud environment, where dynamic IPs are the norm.

Microservices



Load balancer anti-pattern.

Speaker notes

Another common anti-pattern is to front B with a load balancer. This introduces a single point of failure, the load balancer, doubles the number of network hops between services, and increases our costs.

Additional challenges

Clouds & Containers

Speaker notes

Adding to our challenges, is a broad adoption of public clouds and containers. In these environments, we typically have dynamic IPs, higher failure rates, and ephemeral infrastructure that can scale up/down quickly and have a short lifespan. Paired with a complex network topology, our challenges get compounded.

Additional challenges

Clouds & Containers

- Dynamic IP Addresses

Speaker notes

Adding to our challenges, is a broad adoption of public clouds and containers. In these environments, we typically have dynamic IPs, higher failure rates, and ephemeral infrastructure that can scale up/down quickly and have a short lifespan. Paired with a complex network topology, our challenges get compounded.

Additional challenges

Clouds & Containers

- Dynamic IP Addresses
- Higher Failure Rate

Speaker notes

Adding to our challenges, is a broad adoption of public clouds and containers. In these environments, we typically have dynamic IPs, higher failure rates, and ephemeral infrastructure that can scale up/down quickly and have a short lifespan. Paired with a complex network topology, our challenges get compounded.

Additional challenges

Clouds & Containers

- Dynamic IP Addresses
- Higher Failure Rate
- Ephemeral Infrastructure

Speaker notes

Adding to our challenges, is a broad adoption of public clouds and containers. In these environments, we typically have dynamic IPs, higher failure rates, and ephemeral infrastructure that can scale up/down quickly and have a short lifespan. Paired with a complex network topology, our challenges get compounded.

Additional challenges

Clouds & Containers

- Dynamic IP Addresses
- Higher Failure Rate
- Ephemeral Infrastructure
- Complex Network Topology

Speaker notes

Adding to our challenges, is a broad adoption of public clouds and containers. In these environments, we typically have dynamic IPs, higher failure rates, and ephemeral infrastructure that can scale up/down quickly and have a short lifespan. Paired with a complex network topology, our challenges get compounded.

Code Sample

```
service {
  name = "redis"
  port = 8000
  tags = ["global"]
```