Running a distributed system across Kubernetes clusters

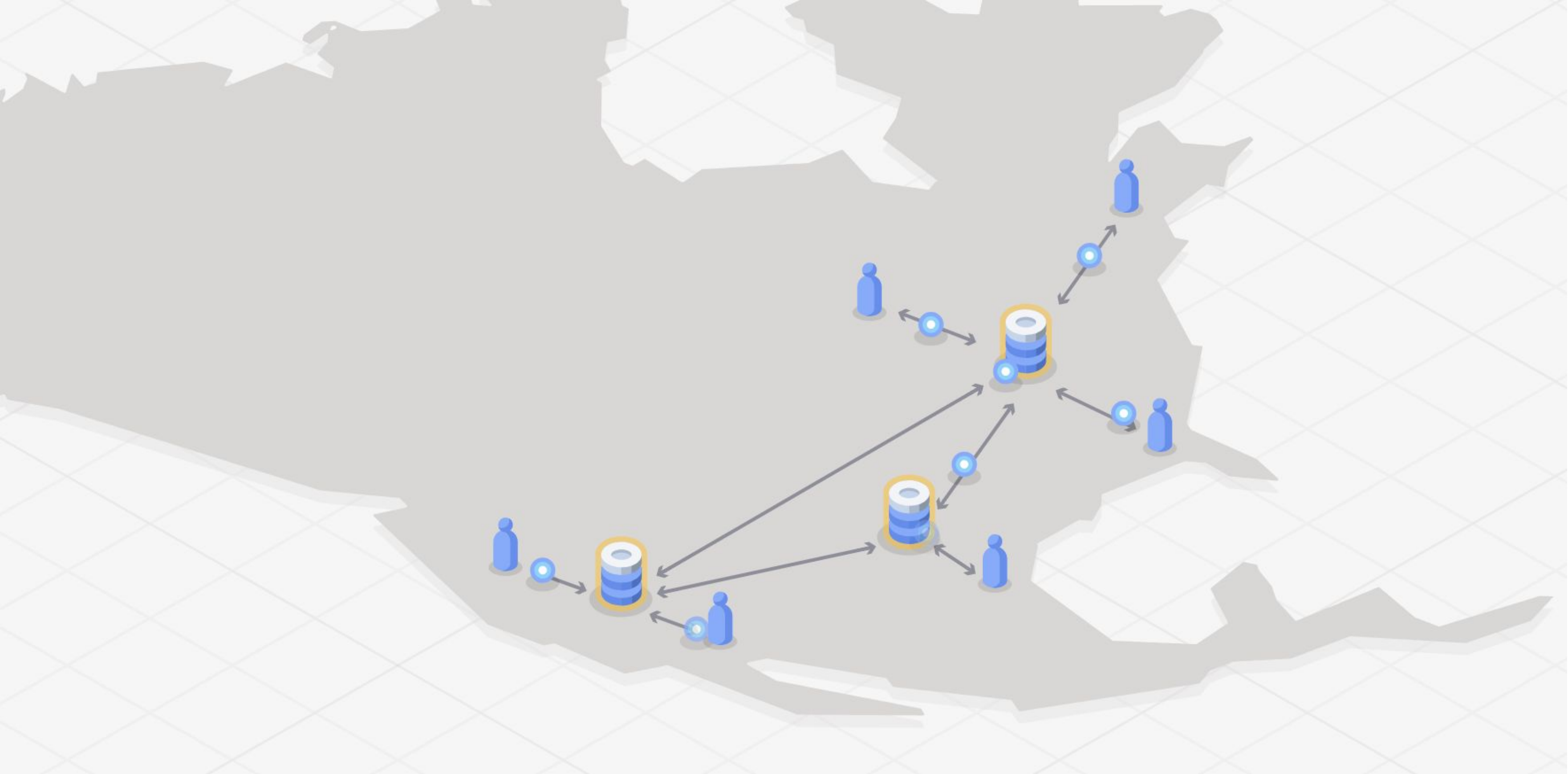
Presented by Alex Robinson / Member of the Technical Staff
@alexwritescode

Cockroach LABS

Running on Kubernetes has gotten much easier

- Dynamic volume provisioning
- Statefulsets
- Multi-zone clusters
- Managed Kubernetes services
- Helm Charts
- Operators

…Unless you want your service to span regions



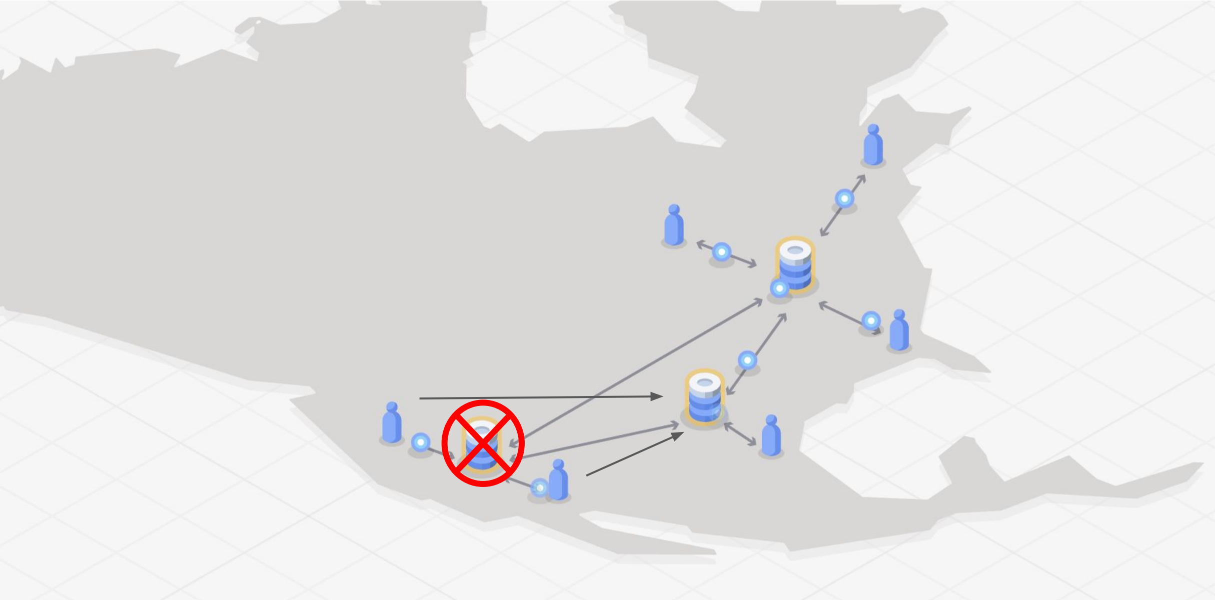
Why run a system across multiple regions?

Why run a system across multiple regions?

- Latency

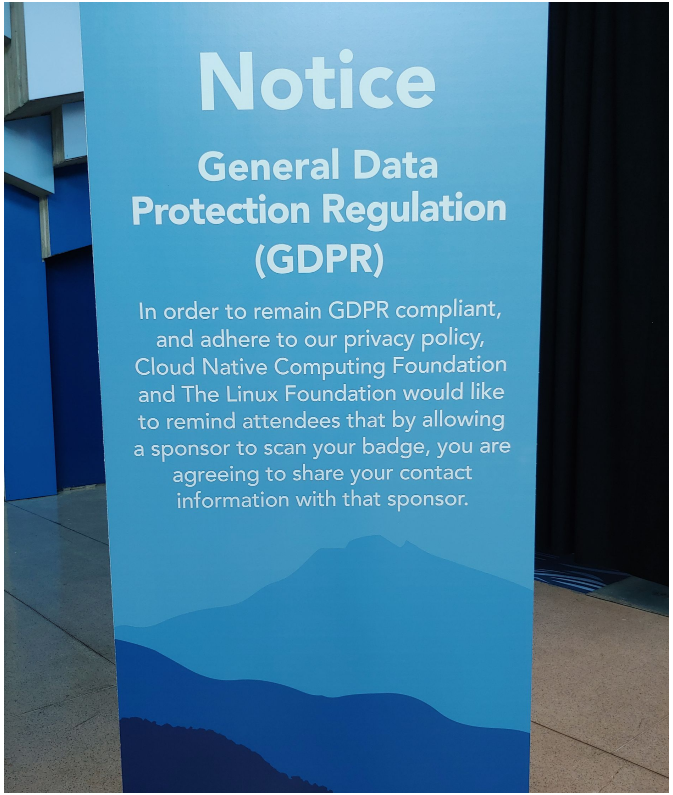
Why run a system across multiple regions?

- Latency
- Fault tolerance



Why run a system across multiple regions?

- Latency
- Fault tolerance
- ...Bureaucracy?



Notice

General Data Protection Regulation (GDPR)

In order to remain GDPR compliant, and adhere to our privacy policy, Cloud Native Computing Foundation and The Linux Foundation would like to remind attendees that by allowing a sponsor to scan your badge, you are agreeing to share your contact information with that sponsor.

Let’s talk about running across Kubernetes clusters

- Why is it hard?
- What do you need to know to get started?
- Solutions

My experience with Kubernetes

- Worked directly on Kubernetes and GKE from 2014-2016
  - Part of the original team that launched GKE
- Lead all container-related efforts for CockroachDB
  - Configurations for Kubernetes, DC/OS, Docker Swarm, even Cloud Foundry
  - AWS, GCP, Azure, On-Prem
  - From single availability zone deployments to multi-region
  - Help users deploy and troubleshoot their custom setups

kubernetes

Cockroach DB

The problem

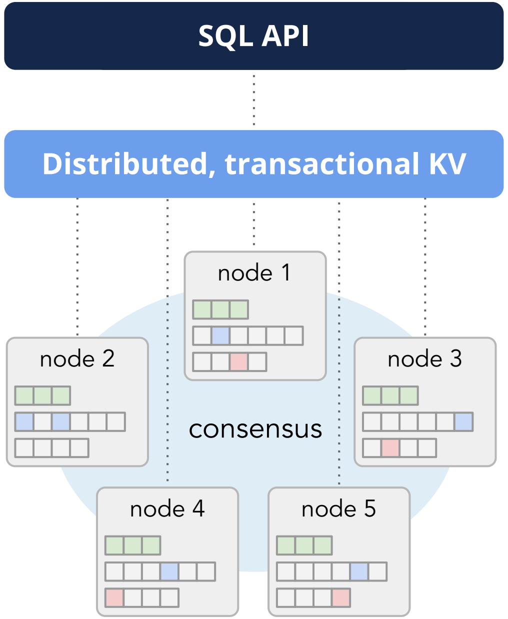
What’s so hard about spanning across clusters?

Multi-region == Multi-Kubernetes-cluster

- Kubernetes is not designed to span WANs
  - Originally didn’t even want to have to span datacenters/AZs within a region, but the community fought for that and made it happen
- Try to run a single k8s cluster across regions at your own risk

What does a distributed stateful system need?

Our example: CockroachDB



CockroachDB is an open source distributed SQL database

1. **SQL:** Your applications use standard PostgreSQL
2. **Ranges:** Tables are sorted by key, split into 64MB chunks and each range is then replicated across the cluster
3. **Nodes:** If a node is added, remove or fails, the cluster automates redistribution and replication of ranges
4. **Consensus:** Consensus protocol ensures consistency and highest level of isolation for transactions
5. **Locality:** In a distributed environment you can tie data to a location (physical, logical, real)

Running CockroachDB in Kubernetes

- Cockroach requires (roughly) three things:
  1. Each node has persistent storage that survives restarts
  2. Each node can communicate directly with every other node
  3. Each node has a network address that survives restarts

Running CockroachDB in Kubernetes

- Cockroach requires (roughly) three things:
    1. Each node has persistent storage that survives restarts
    2. Each node can communicate directly with every other node
    3. Each node has a network address that survives restarts
- Kubernetes provides these very well within a single cluster

Running CockroachDB in Kubernetes

- Cockroach requires (roughly) three things:
  1. Each node has persistent storage that survives restarts
  2. Each node can communicate directly with every other node
  3. Each node has a network address that survives restarts
- Kubernetes provides these very well within a single cluster
- Across multiple Kubernetes clusters, we lose #3 and often #2

The core problems: what's missing?

- It all comes down to networking:
    - Pod-to-pod communication across clusters
        - Including across private networks (e.g. cloud VPCs) when applicable
    - Persistent address that works both within and between clusters

Kubernetes networking

What’s the deal with multi-cluster networking?

Kubernetes networking

- Kubernetes doesn't care *how* it's done, but it requires that:
  1. Each pod has its own IP address
  2. The IP that a pod sees itself as is the same IP that others see it as
  3. All pods can communicate with all other pods without NAT

Kubernetes networking

- Kubernetes doesn’t care *how* it’s done, but it requires that:
  1. Each pod has its own IP address
  2. The IP that a pod sees itself as is the same IP that others see it as
  3. All pods can communicate with all other pods without NAT
- The problem: we want to run tens of pods on each machine, but traditional networks only allocate one or two IP addresses to each host

Kubernetes networking

- From the very beginning, Kubernetes made these specific demands about what the network in a cluster must enable
    - Since then, dozens of solutions have been built to satisfy those requirements
- But the multi-cluster scenario is left completely unspecified

Google Cloud
Azure
weaveworks
cilium
CISCO
PROJECT CALICO
amazon web services
flannel

Kubernetes networking

- Some enable pod-to-pod communication across clusters out of the box
    - e.g. GKE, EKS
- Some make it fairly easy to enable
    - e.g. AKS (“advanced” networking mode), Cilium (“cluster mesh”)
- It can be quite difficult in others

Solutions

So what can we do?

Multi-cluster solutions

- The options all have one problem or another
    - Don’t work in subsets of environments
    - Break easily if operator doesn’t know exactly what they’re doing
    - Use a slow, less reliable datapath
    - Don’t work with TLS certificates
    - Rely on a relatively immature, complex system

Solution #1: Static pod placement + HostNetwork

- `HostNetwork` option lets a pod use the host machine's network directly
- Use each host's routable IP address for all communication
- Statically assign pods to machines so that all nodes' IPs stay the same

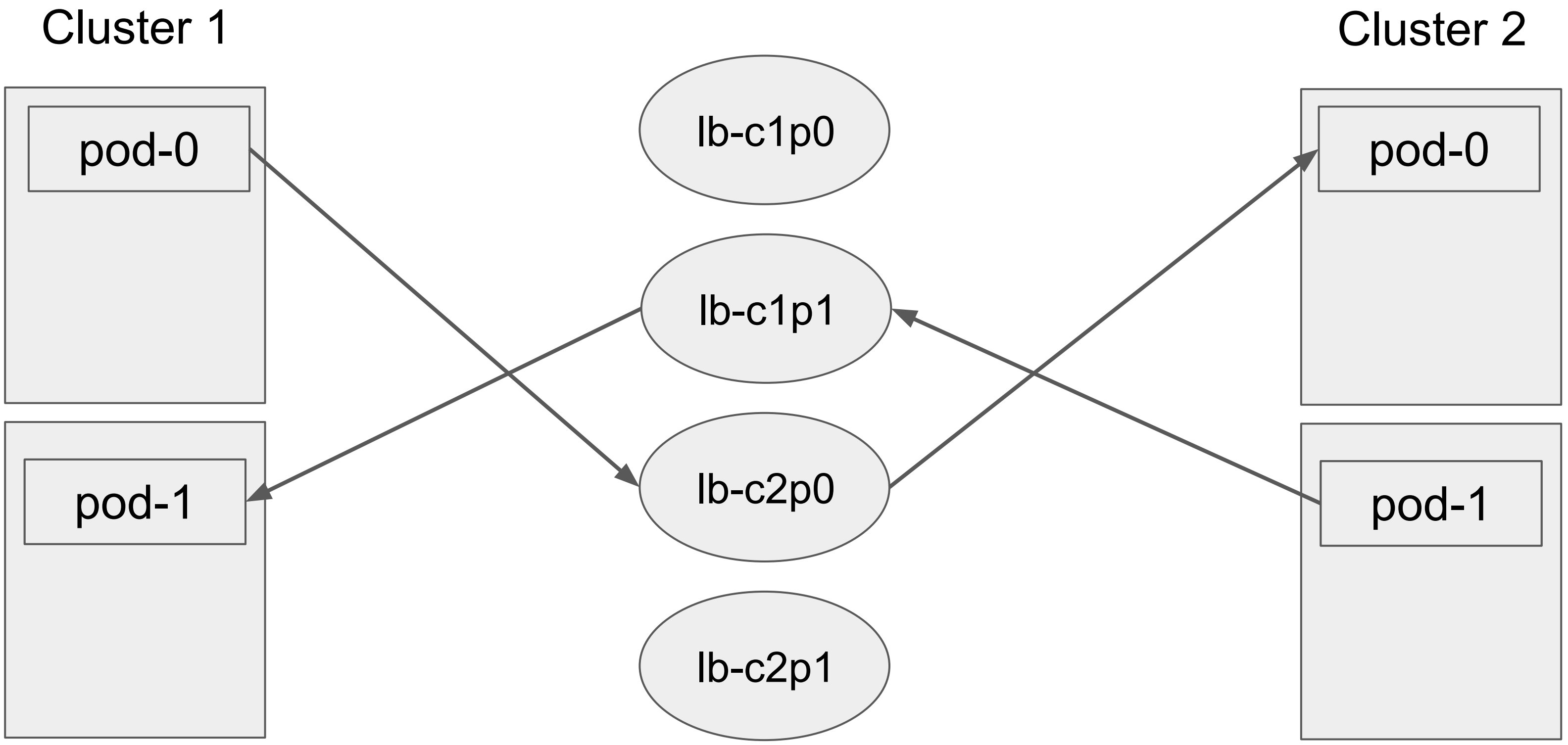
Solution #1: HostNetwork + Public IPs

- Pros:
  - Works even when pod-to-pod communication between clusters doesn’t
  - No moving parts
  - Using `HostNetwork` can actually give a nice performance boost
- Cons:
  - Depends on host IPs not changing
    - Some cloud providers delete and recreate VMs during node upgrades
  - Requires a lot of manual config-file editing and certificate creating
  - Uses up valuable ports on the host machines, can hit port conflicts

Solution #2: External load balancer for each pod

- Create a public load balancer for each Cockroach pod
    - It’s really easy to expose a service with a load balancer in all the major cloud providers
- Have all pods connect to each other with the load balancer IPs/DNS names

Solution #2: External load balancer for each pod



Solution #2: External load balancer for each pod

- Pros:
  - Works even when pod-to-pod communication between clusters doesn’t
  - Continues working even as Kubernetes hosts churn in/out of the cluster
  - No need to configure a cross-cluster naming service
    - Because the load balancer addresses never change once they’re created

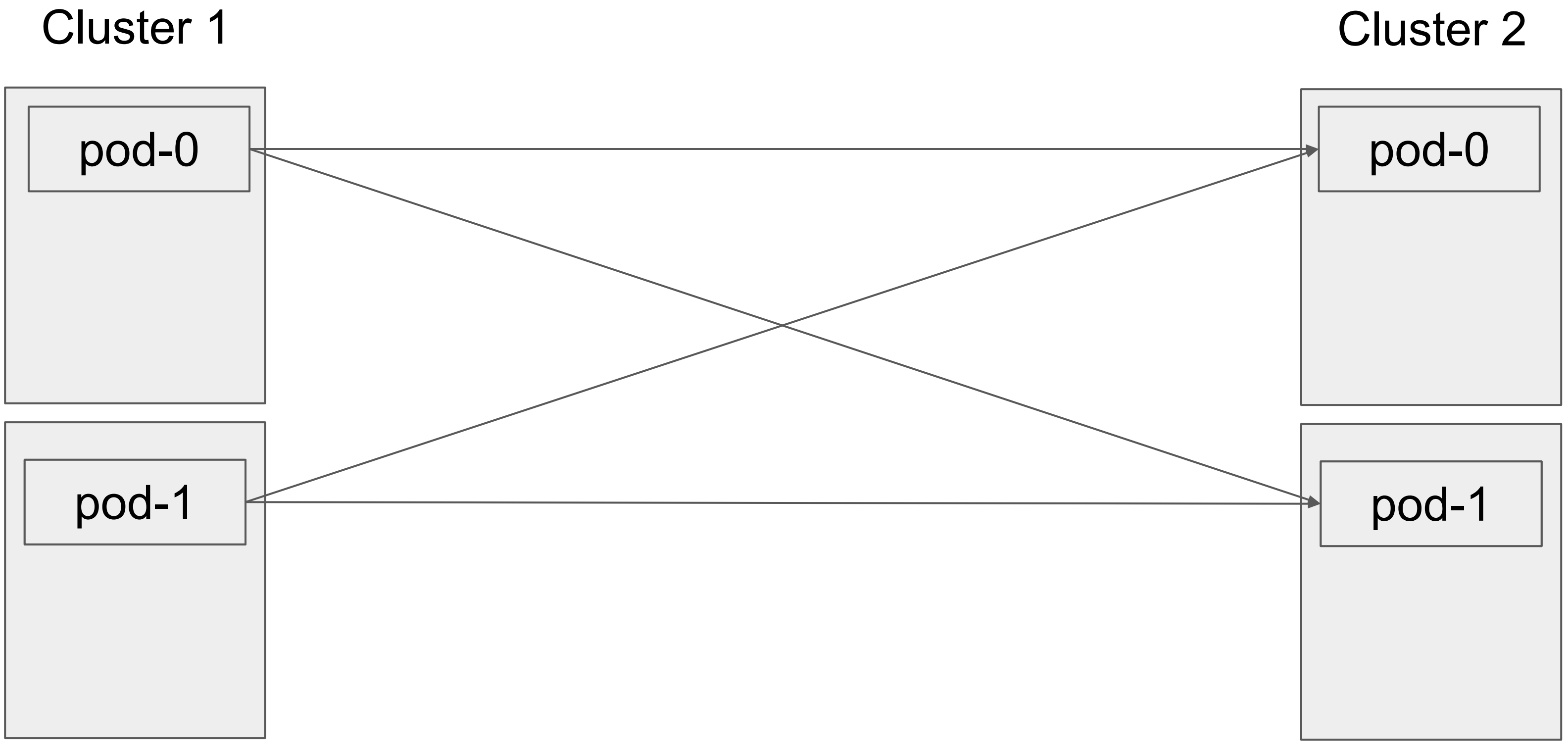
Solution #2: External load balancer for each pod

- Cons:
    - Requires provisionable load balanced addresses - not always available on-prem
    - Can be expensive to run so many load balancers
    - A lot of manual config-file editing and cert creating with LB addresses
        - Informing each pod of its public IP requires pretty complex configuration
    - Need to create new service and LB whenever you want to scale up
    - Extra hops on data path on all but the most sophisticated load balancers
        - src pod->src host->LB->random host->dst host->dst pod

Solution #3: Use pod IPs directly

- Just let pods advertise their pod IPs to each other directly
    - Assuming pod-to-pod connectivity across clusters
- Rely on Cockroach's handling of address changes to deal with the inevitable churn as pods move to different nodes and get different IPs
- Set up an internal load balancer that only gets used for the --join flag
    - Won't be used on the data path, since cockroach nodes will share their pod IP addresses internally via gossip once joining

Solution #3: Use pod IPs directly



Solution #3: Use pod IPs directly

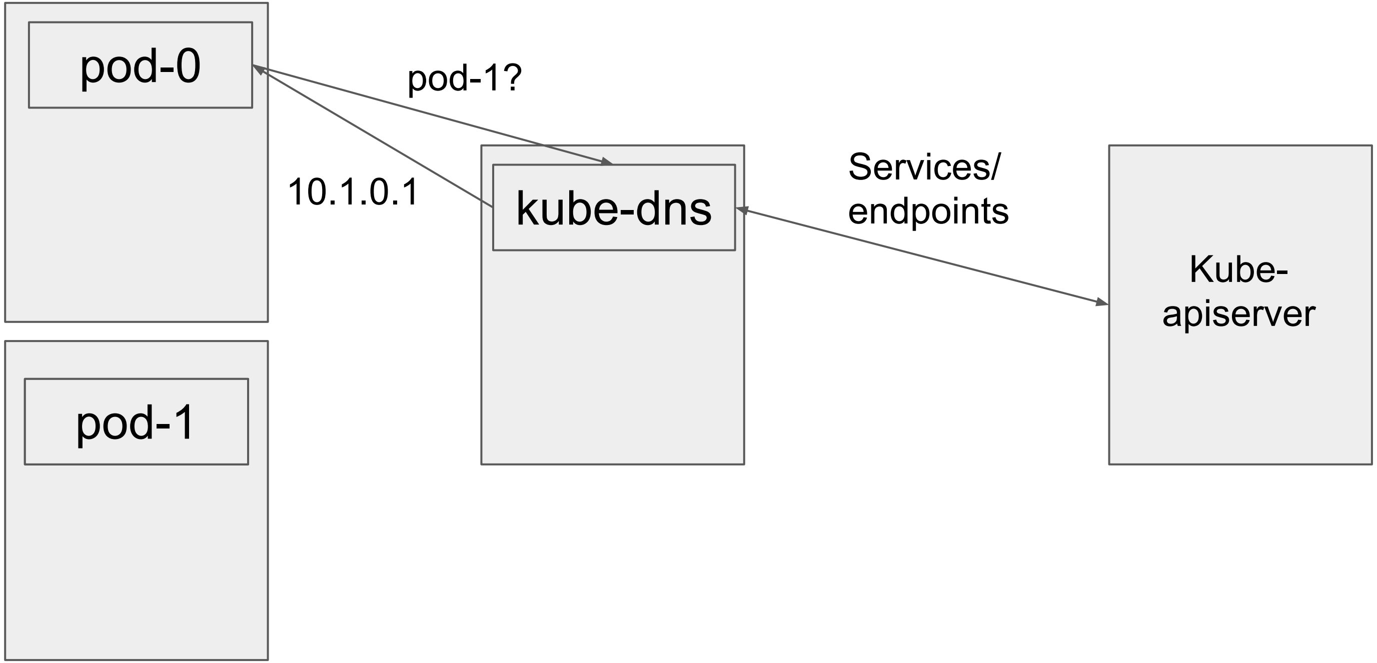
- Pros:
  - No overhead on data path (except normal Docker overhead)
  - Very resilient to changes like hosts being removed/added/restarted
  - Very little manual work needed to configure and maintain
- Cons:
  - Network must support direct pod-to-pod communication across clusters
  - Because IP addresses can change across pod deletions/recreations, creating TLS certificates that can stand up to hostname verification is very tricky

Solution #4: DNS chaining

- Basically: how can we add persistent names to the previous solution?
    - Needed for TLS certificates or for systems that can’t handle changing addresses

Solution #4: DNS chaining

Cluster 1



Solution #4: DNS chaining

- Option 1: Use CoreDNS instead of the default kube-dns service
  - CoreDNS allows for much more customization, including plugins that allow it to watch multiple Kubernetes apiservers
    - https://github.com/coredns/kubernetai was written to do just what we want
  - But swapping CoreDNS in for kube-dns on managed offerings is shockingly difficult
    - As is customizing each cluster's DNS domain from the `cluster.local.` default, which would be important for this approach
  - CoreDNS is becoming the standard as of 1.13, so this should become more feasible as more clusters switch over to it

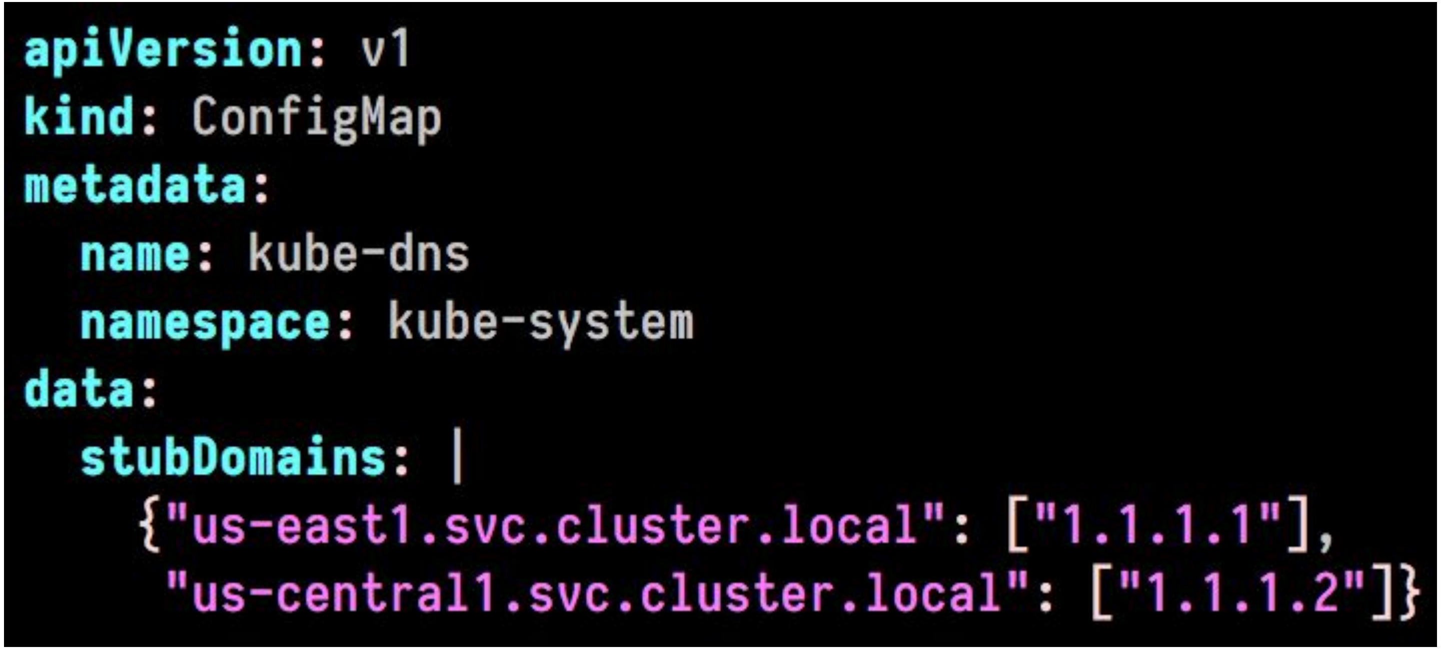
Solution #4: DNS chaining

- Option 1.5: Use CoreDNS alongside the default kube-dns service
  - Configure kube-dns to defer certain lookups to CoreDNS
  - Configure CoreDNS to watch the other k8s clusters’ apiservers
  - Add in some CoreDNS rewrite rules to make cross-cluster lookups work out
  - Haven’t properly tried this out so I’m being a little fuzzy on the details

Solution #4: DNS chaining

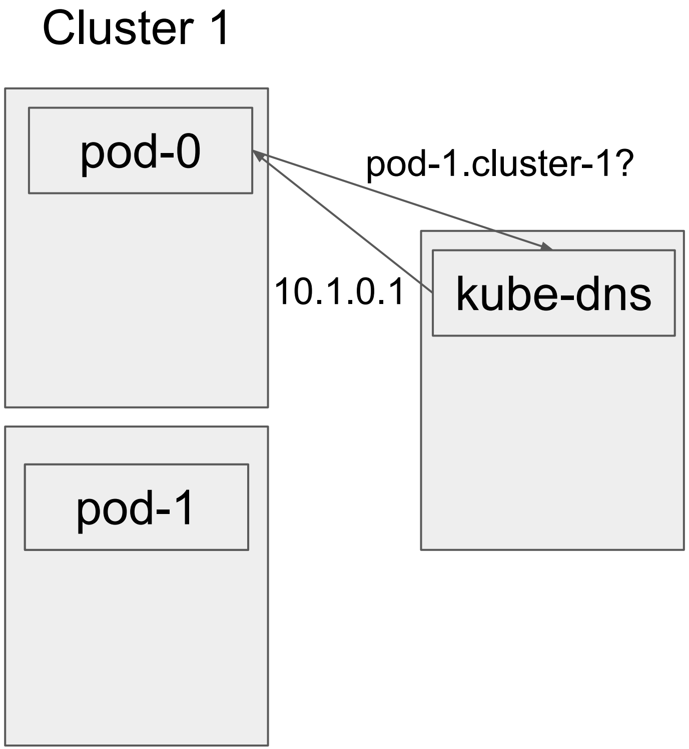
- Option 2: Chain DNS servers together using “stub domains”
    - Use KubeDNS config feature to defer DNS lookups for certain domains to the nameserver of your choice
    - e.g., configure KubeDNS in us-west1 to redirect lookups for `*.us-east1.svc.cluster.local` to the us-east1 cluster’s DNS service
    - This is what I’ve put together scripts for people to try out on GKE

Solution #4: DNS chaining

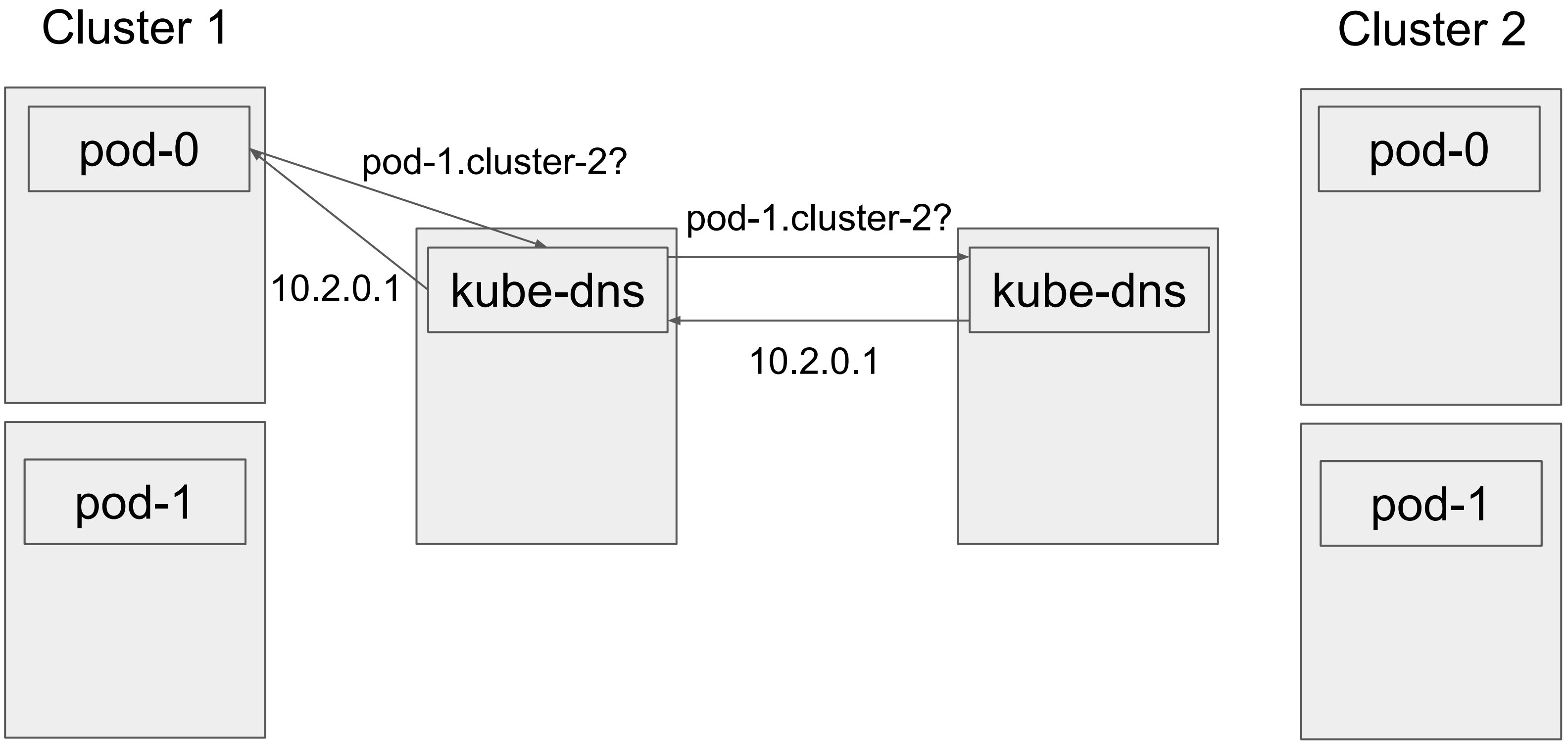


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
data:
  stubDomains: |
    {"us-east1.svc.cluster.local": ["1.1.1.1"],
     "us-central1.svc.cluster.local": ["1.1.1.2"]}
```

Solution #4: DNS chaining



Solution #4: DNS chaining



Solution #4: DNS chaining

- Pros of using stub domains:
  - No overhead on data path (except normal Docker overhead)
  - Very resilient to changes like hosts being removed/added/restarted
  - No need to add any extra controllers to the clusters
  - Very little manual work needed to get running (totally scriptable)
- Cons of using stub domains:
  - Network must support pod-to-pod communication across clusters
  - Need to set up load balanced endpoints for each DNS server
    - Cloud support for this isn't great…

Solution #5: Istio

- Istio has been working on a “multi-cluster” mode to handle the problem of addressing services across clusters
  - It’s explicitly not addressing the problem of pod-to-pod connectivity, though -- just naming
- Install Istio control plane in one primary k8s cluster, then install special “istio-remote” components in others

Solution #5: Istio

- Pros:
  - Under very active development, looks likely to improve over time
  - Small overhead on data path (packets go through Envoy proxy)
  - Very resilient to changes like hosts being removed/added/restarted (at least in theory)
- Cons:
  - Still immature - entire control plane runs in a single k8s cluster (single point of failure)
  - Very involved setup process
  - Docs say “production environment might require additional steps”, at least one of which essentially boils down to solving this problem for the Istio components themselves
  - Requires copying k8s cluster credentials into each other (potential security concern)

Solution #6: Roll your own

- Can always do your own thing, e.g.:
  - Set up your own custom DNS servers outside Kubernetes
  - Set up your own auto-certificate approver for pod IPs then use them directly
  - Manage your own clusters, use CoreDNS, and modify it as you please
  - Set up your own proxy server(s) to manage stable, routable addresses
  - Etc.
- Tough for us to widely recommend due to environmental differences, but actually quite reasonable if you have a single stable environment

Summary

Conclusions

- Multi-cluster networking is a bit of a minefield
  - Left completely unspecified, so different installations can vary wildly
  - Very little has been done about it for years
- Even after you nail pod-to-pod connectivity, you still have to solve naming
- People are finally starting to care, though! (e.g. Cilium, Istio, Upbound)
- It’s hard to recommend a single answer for everyone, but there are very reliable options if you’re willing to spend some time up-front on setup

Thank You!

For more info:

cockroachlabs.com

github.com/cockroachdb/cockroach

@alexwritescode

Cockroach LABS

Questions?

Cockroach LABS