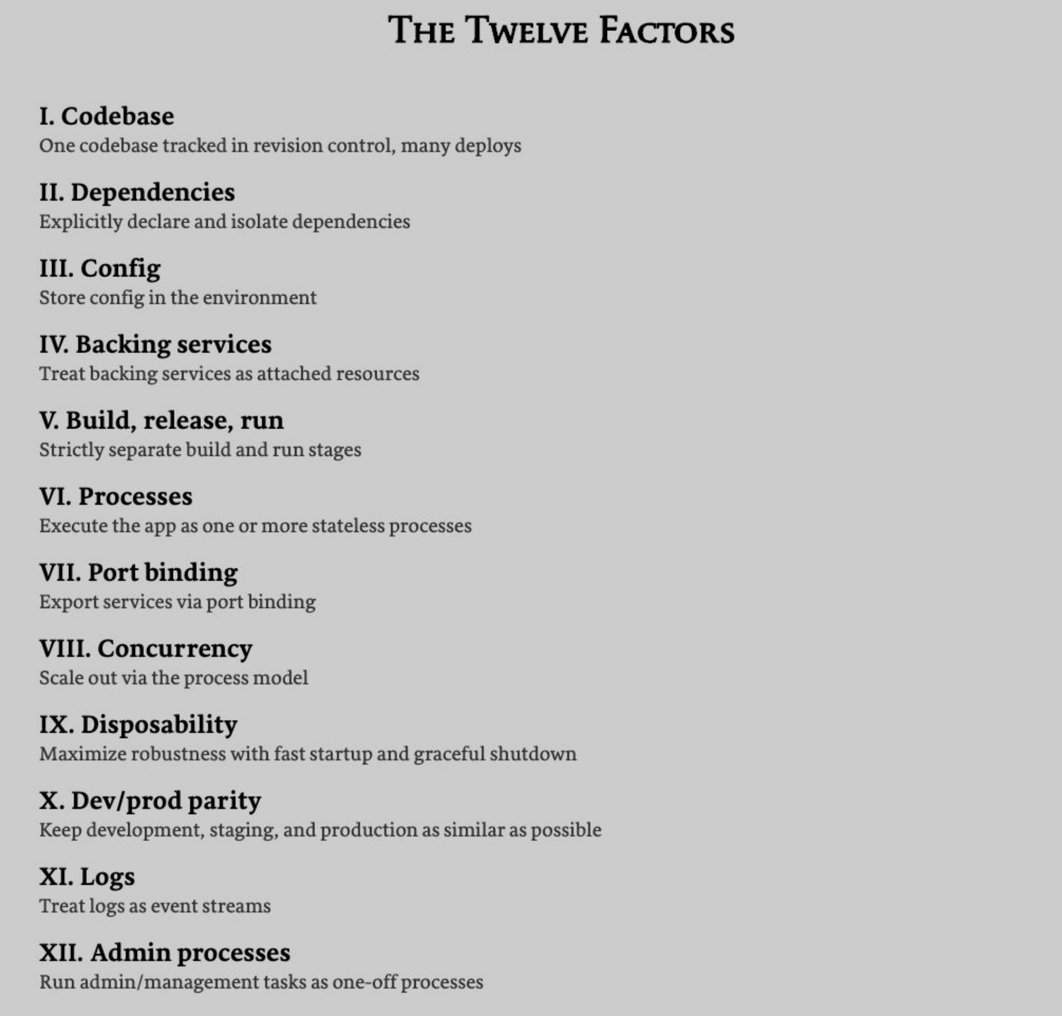
# CockroachDB

Distributed SQL at scale

Cockroach LABS



## The Twelve Factors

**I. Codebase**
One codebase tracked in revision control, many deploys

**II. Dependencies**
Explicitly declare and isolate dependencies

**III. Config**
Store config in the environment

**IV. Backing services**
Treat backing services as attached resources

**V. Build, release, run**
Strictly separate build and run stages

**VI. Processes**
Execute the app as one or more stateless processes

**VII. Port binding**
Export services via port binding

**VIII. Concurrency**
Scale out via the process model

**IX. Disposability**
Maximize robustness with fast startup and graceful shutdown

**X. Dev/prod parity**
Keep development, staging, and production as similar as possible

**XI. Logs**
Treat logs as event streams

**XII. Admin processes**
Run admin/management tasks as one-off processes

The underlying philosophy for microservices is a **12 factor app.**

12factor.net

**Factor Number VI (Processes):** Make your microservices stateless.

✓

Our micro-services should be **Cattle**

✗

not **Pets**

But…

Where should we keep our data?

Traditional databases are single instance: if it goes down, YOU go down.

And if you want to scale it, you have to build that single instance taller and taller.

Not horizontally, like our microservices.

"I'VE LOOKED AT THE SYSTEM AND FOUND A SINGLE POINT OF FAILURE. YOU."

And…
single-instance is a SPOF.

To get resiliency and scalability, you can use a **NoSQL** Database...

But can it be **SQL**?

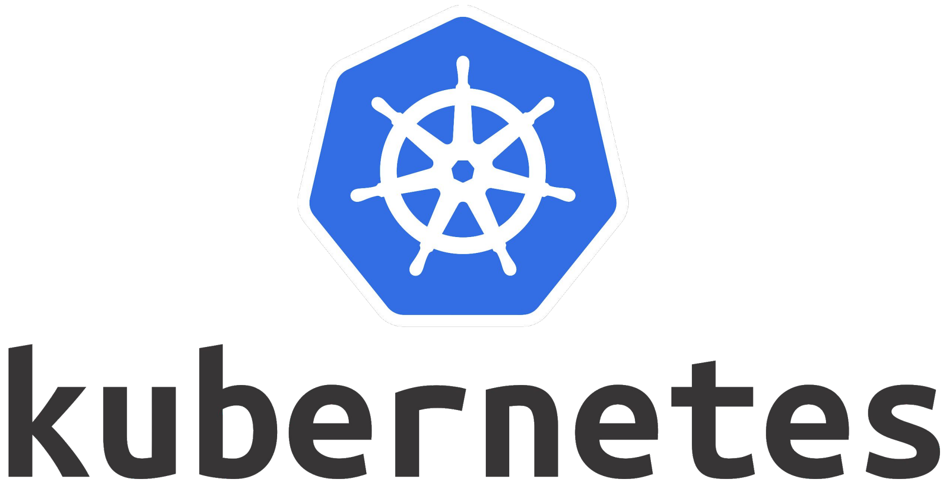
**Distributed SQL**
blends old and new:

- NoSQL scalability
- SQL compatible
- ACID guarantees

ATOMICITY CONSISTENCY ISOLATION DURABILITY

The usual suspects have cloud offerings.

But this also needs to be solved across clouds or in our own labs and datacenters to avoid cloud lock-in.



Most importantly, it should run inside K8s.

CockroachDB turns SQL into a resilient, scalable, ACID-compliant cluster.

# Why name it CockroachDB?

# CockroachDB in a sentence...

Cockroach DB

CockroachDB clusterizes a single-instance SQL database so it can survive failures, scales, yet act as if it’s a single, logical database.

# What CockroachDB Solves

## Examples of what CockroachDB can solve...

1. Provide a resilient datastore for microservices which runs in Kubernetes.
2. Host transactional, always-on database workloads.
3. Provide systems of record (ledger, transactional backends, & more.) or systems of access (such as identity access systems.)
4. Run multi-region databases with immediate consistency (full ACID guarantees).
5. Meet GDPR requirements by choosing where to domicile data.

# Production Examples

## CockroachDB in Production

1. **Telecom provider**: Customer service virtual agent and chat.
2. **Major Bank**: Core services in a multi-region, resilient cluster.
3. **European badging system manufacturer**: Access management system requiring low-latency reads and immediate consistency.

# Production Examples

## CockroachDB in Production

4. **Financial Company**: Customer Identity Access Management with immediate consistency for a company that provides many online products with a single login. Prior replicated system had production-impacting time lags.

5. **Gaming Company**: Primary transactional database for financial transactions across all of Europe while adhering to the GDPR regulations.

# CockroachDB Characteristics

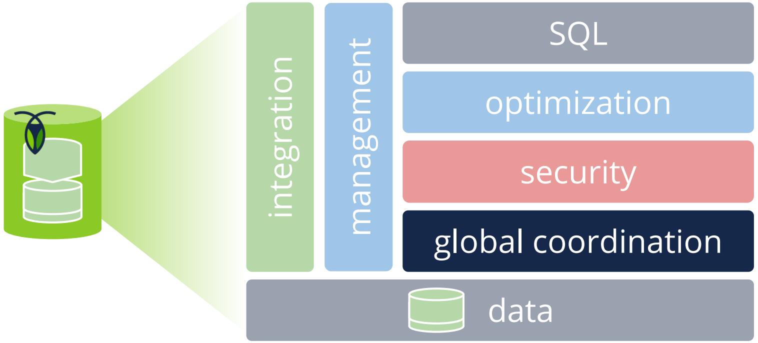
### Characteristics

1. **OLTP Database** best for transactional workloads.
2. **ANSI SQL Postgres** wire-compatible (9.6).
3. **ACID Compliant** with Serializable Isolation.
4. **Postgres Driver** connect to any node to connect and your code sees a single DB.
5. **Horizontally scalable** by adding nodes whenever capacity or resiliency is needed.
6. **Kubernetes friendly** by just installing a cluster with a helm chart. To scale, just add instances.

# CockroachDB: a unique distributed architecture

**self contained, aware nodes** participate in global cluster

Each node within a cluster is self-contained and has locational awareness of their self and others



- Every node is a CONSISTENT gateway to the entire database

- Intelligence packed with data
  - ○ Management & optimization
  - ○ Standard SQL engine
  - ○ Enterprise security
  - ○ Ecosystem integration

# CockroachDB: a unique distributed architecture

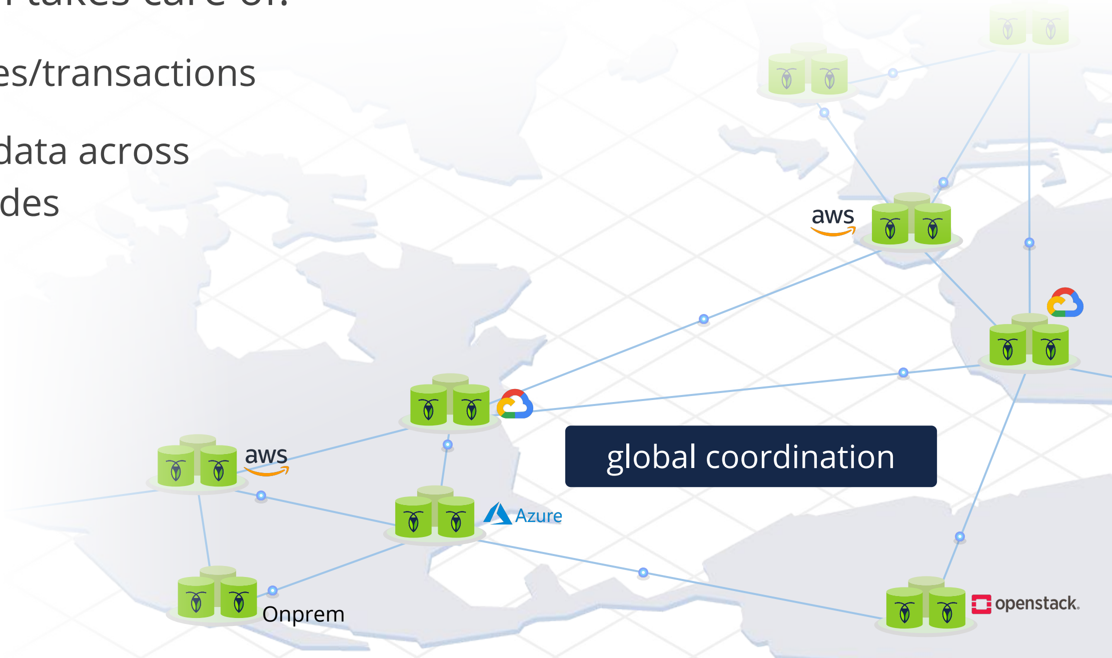
**global database cluster coordination** and logic

Spin up a node anywhere (public and private clouds) and then point it at the cluster, which takes care of:

- Coordination & consensus for queries/transactions
- Replication, repair & rebalancing of data across cluster upon addition/removal of nodes
- Attach location to any data to set domiciling & replication constraints

Inherently **multi-cloud**

aws Azure openstack.

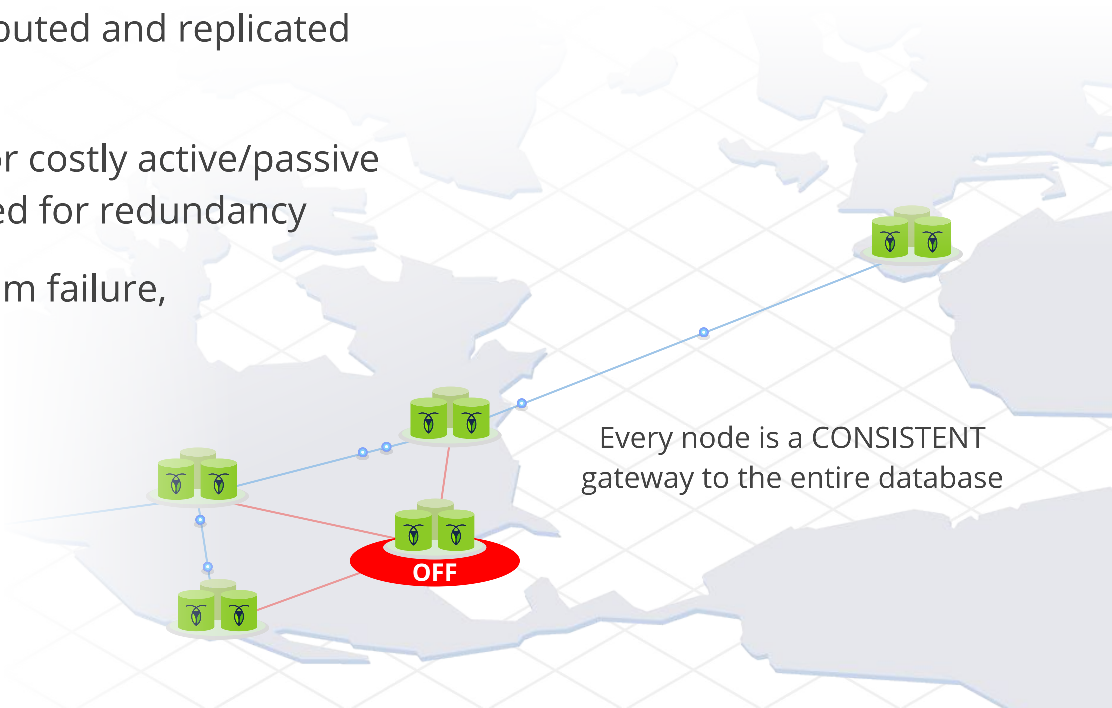


Cockroach LABS

# CockroachDB: Always on and naturally **resilient**

## Your data is always on and always available

1. On failure, data is efficiently redistributed and replicated across nodes within clusters
2. CockroachDB eliminates the need for costly active/passive or complex CDC architectures needed for redundancy
3. Minimize impact & recovery time from failure, with Cockroach RTO is zero



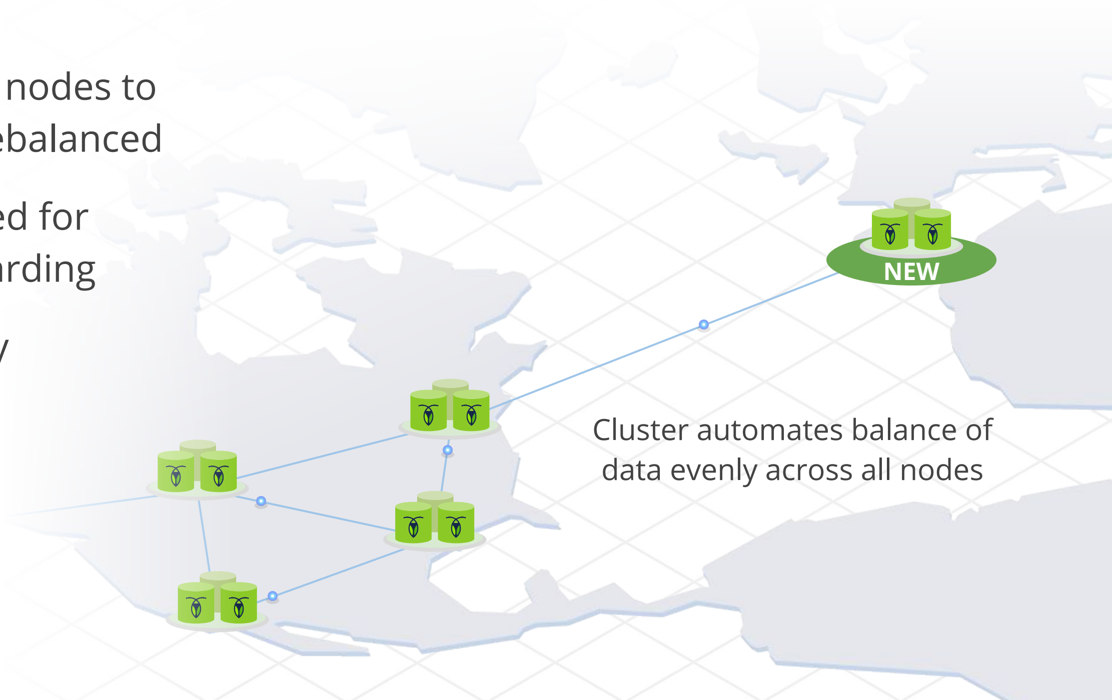
Every node is a CONSISTENT gateway to the entire database

Cockroach LABS

# CockroachDB: **Scale** your data not your complexity

## Replication, repair & rebalancing of data across cluster upon addition or removal of nodes

1. To expand capacity, simply add new nodes to the cluster & data is automatically rebalanced
2. Automated balancing eliminates need for manual sharding and complex resharding
3. Balancing optimizes server efficiency (storage and compute)



Cluster automates balance of data evenly across all nodes

Cockroach LABS

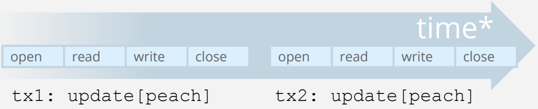
# CockroachDB: Guarantees global **consistency**

## Ensures consistency across distributed transactions

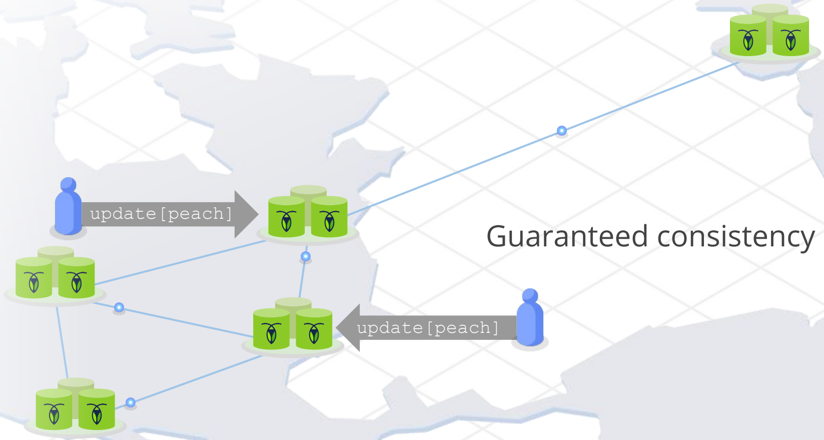
**CockroachDB** uses clocks and concurrency controls to deliver **full ACID** transactions at scale even in a distributed environment

Serializable isolation protects from write skew and dirty reads

CockroachDB: Serializable isolation in a distributed SQL database



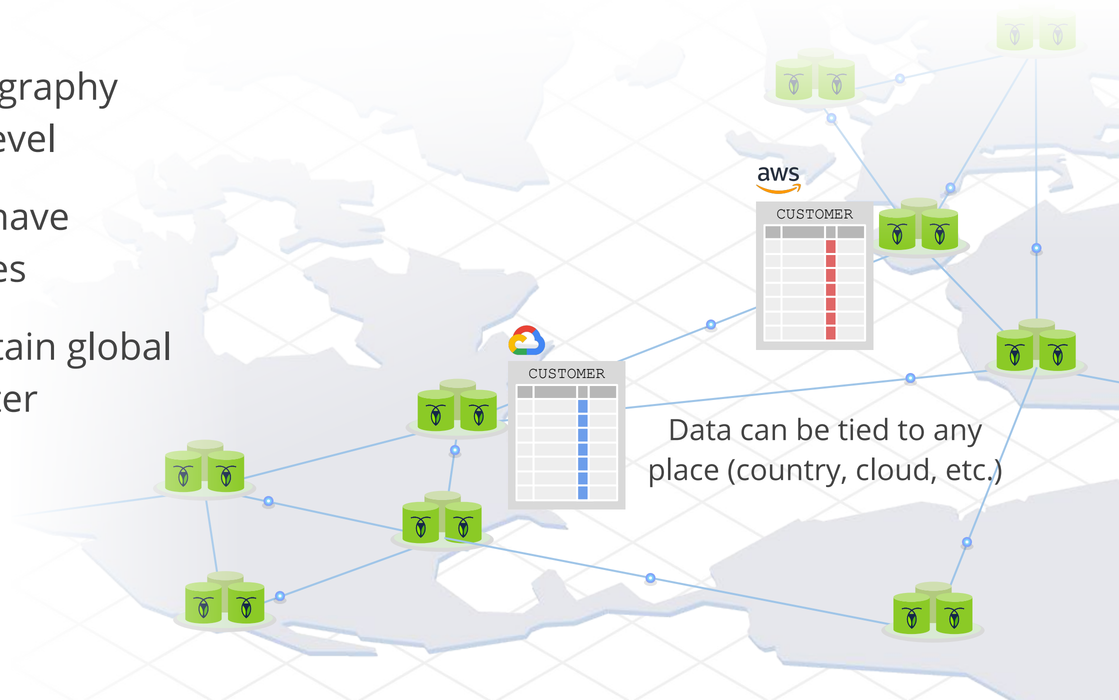
*transactions may not physically execute serialized in time, rather they execute as if they have. They are guaranteed to appear serialized



# CockroachDB: Tie your data to a **location**

Geo-partition your data to set domiciling & replication constraints

1. Tie explicit “ranges” of data to a geography or any address at the table or row level
2. Comply with privacy regulation OR have data follow a user to reduce latencies
3. Tie data to explicit clouds and maintain global access to all nodes throughout cluster

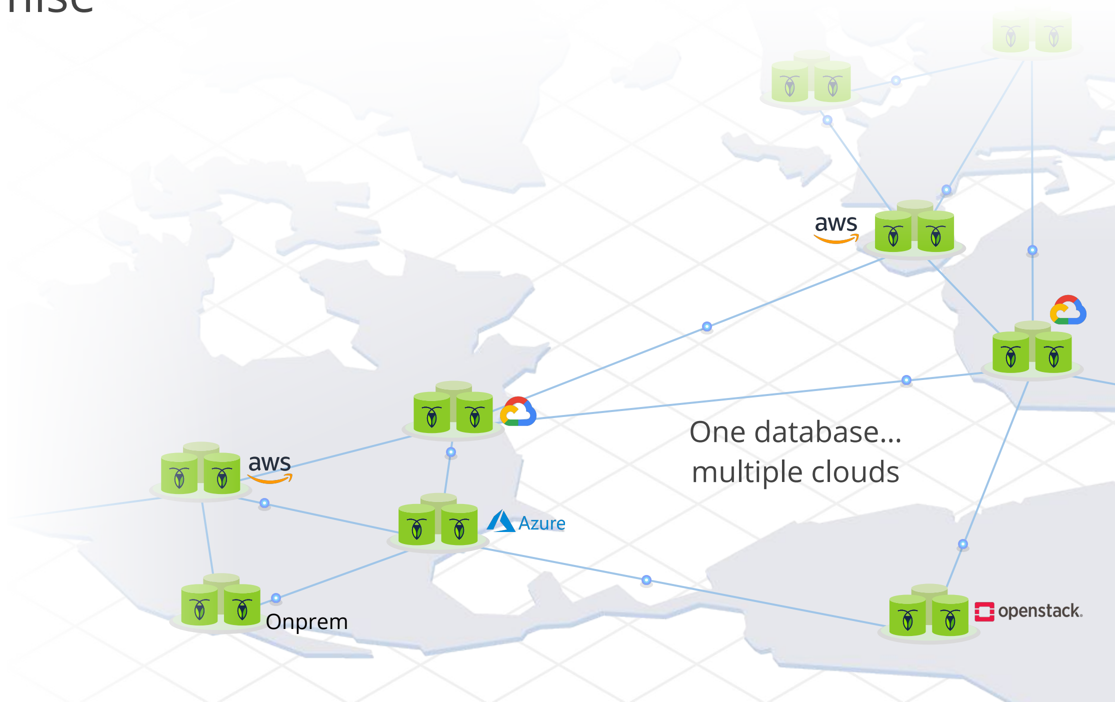


Data can be tied to any place (country, cloud, etc.)

Cockroach LABS

# CockroachDB: Inherently **multi-cloud**

Implement a globally consistent database across clouds and even on premise



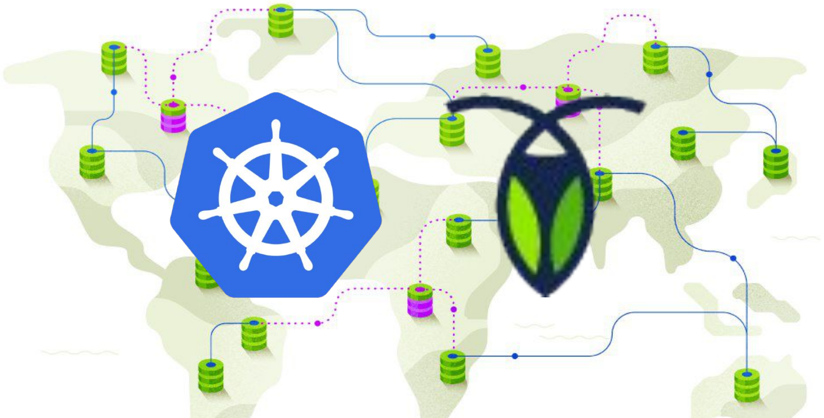
One database...
multiple clouds

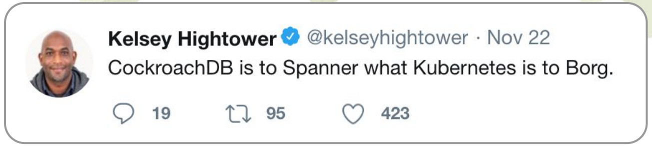
Cockroach LABS

# CockroachDB and Kubernetes

## Common distributed architecture

- Natural fit for pods and orchestration
- Helm chart available
- Multi-region and globalscale
- Geotagging within CRDB helps tie compute to data and locality





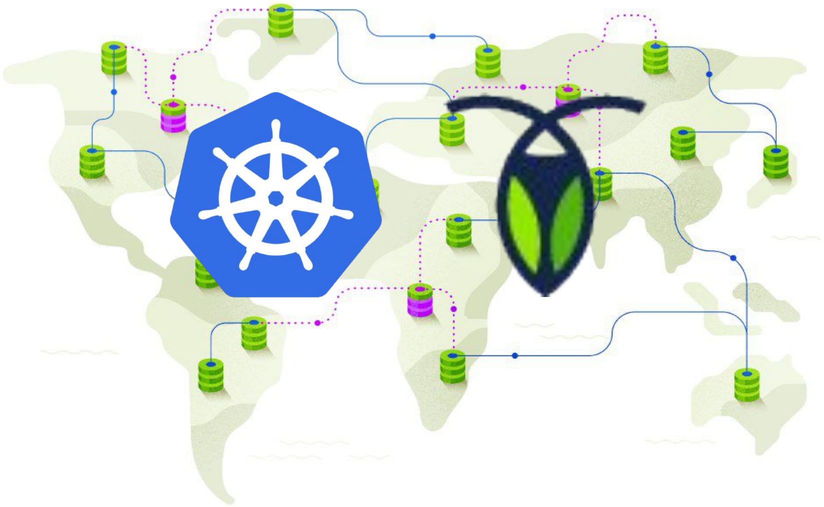
**Kelsey Hightower** @kelseyhightower · Nov 22
CockroachDB is to Spanner what Kubernetes is to Borg.
19 95 423

CockroachDB uses the Storage class and PV claim to mount a volume within a cluster and then builds on stateful sets, so we naturally inherit the controls and power of Kubernetes

# Installing CockroachDB on K8s

## K8s Installation

- Helm chart or customize the YAML on K8s 1.8+.
- Uses a PV with Stateful Sets.
- Runs best with SSDs.



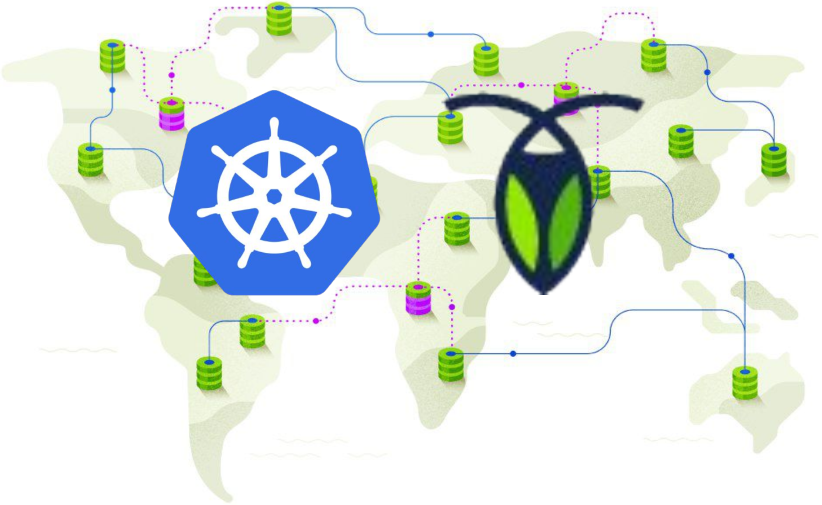
bit.ly/crdbk8s

# Installing CockroachDB on K8s

## K8s Installation

bit.ly/crdb-k8ideas

bit.ly/crdb-docker-ideas

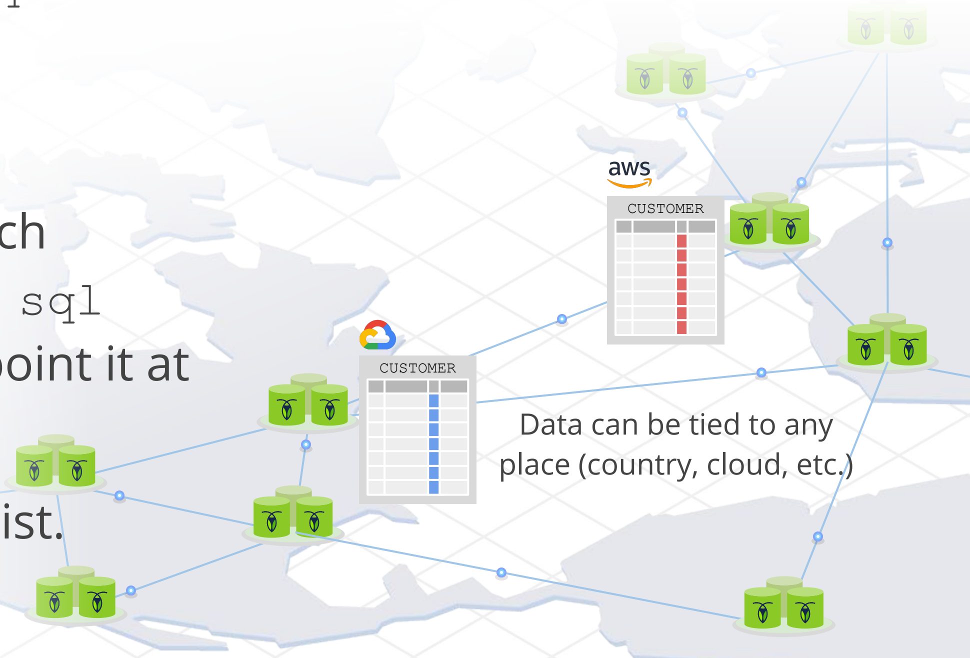


https://github.com/timveil-cockroach/kubernetes-examples
https://github.com/timveil-cockroach/docker-examples

# Connecting to CockroachDB

Connection options for CockroachDB

1. With the SQL shell. (`cockroach sql` from the command line.)
2. With jdbc (jdbc.postgresql.org)
3. Within Kubernetes. (Use a cockroach container and run the `cockroach sql` command and set the `--host` to point it at the internal vip.)
4. See **bit.ly/crdb-connect** for a full list.



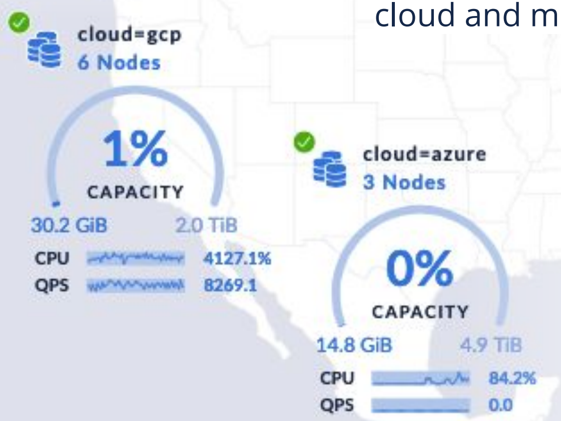
Data can be tied to any place (country, cloud, etc.)

# CockroachDB

Cloud native distributed SQL for the cloud native future

## Cloud Neutral

Build across on-prem, cloud, hybrid cloud and multi cloud environments



## Open Source

UN-opinionated and community driven so you are not tied to any cloud

DOWNLOAD NOW!

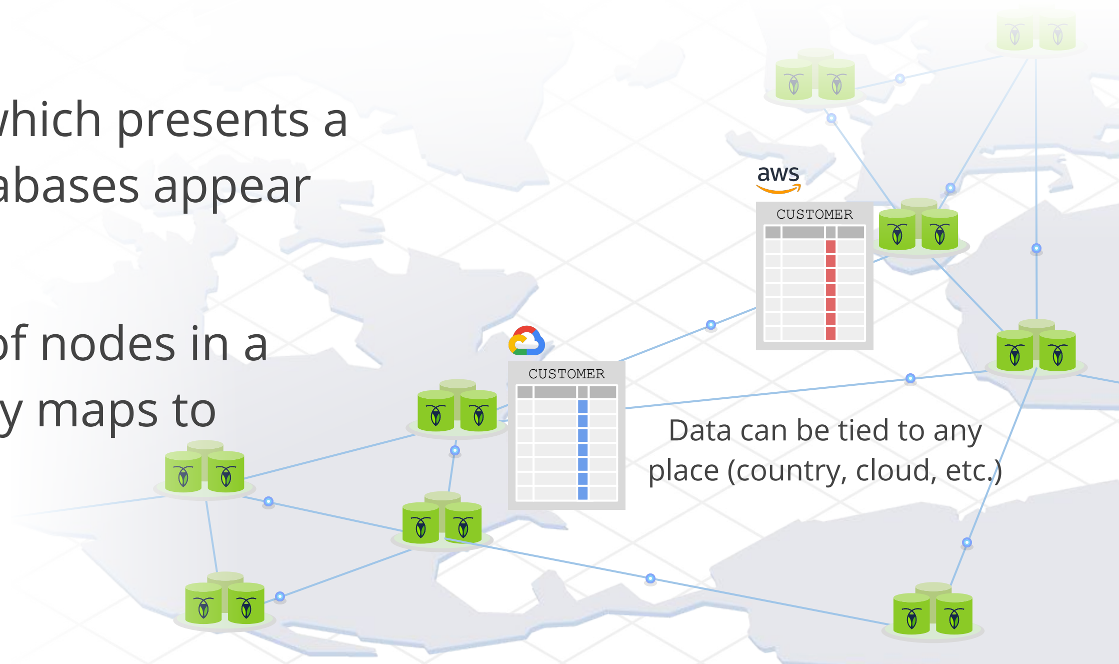
Cockroach LABS

# CockroachDB Architecture Overview

Cockroach LABS

# CockroachDB Cluster concepts

1. **Node**: An individual machine running CockroachDB. Many nodes join together to create your cluster.
2. **Cluster**: A group of nodes which presents a single DBMS engine. All databases appear as single logical instances.
3. **Replication Zone**: Groups of nodes in a single location, which usually maps to regions.

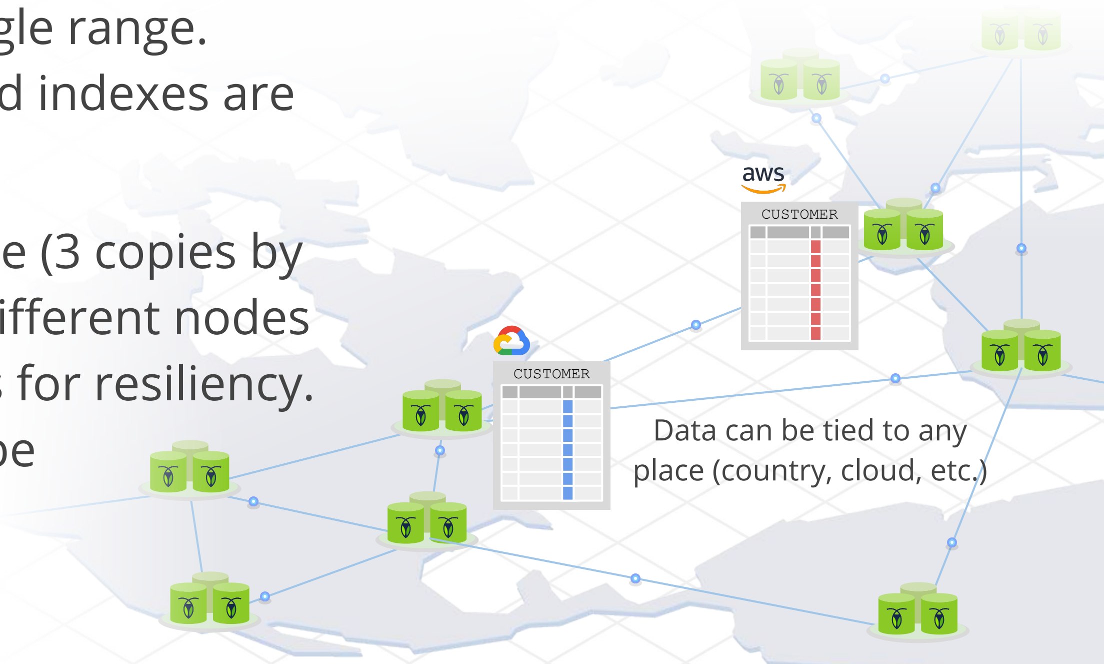


Data can be tied to any place (country, cloud, etc.)

Cockroach LABS

# CockroachDB Data concepts

1. **Range**: User Data is stored in ranges, contiguous chunks of data so that every key can always be found in a single range. (64MB by default.) Tables and indexes are spread across ranges.
2. **Replica**: A copy of each range (3 copies by default) which is stored on different nodes in different replication zones for resiliency. The **replication factor** can be configured.

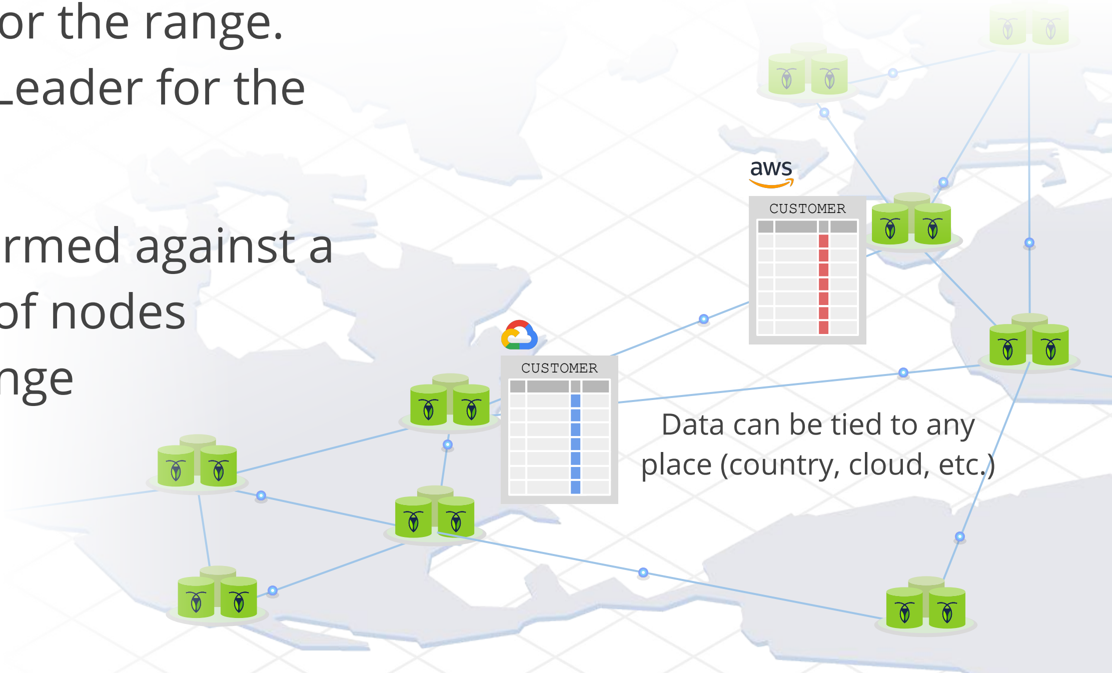


Data can be tied to any place (country, cloud, etc.)

# CockroachDB Data concepts

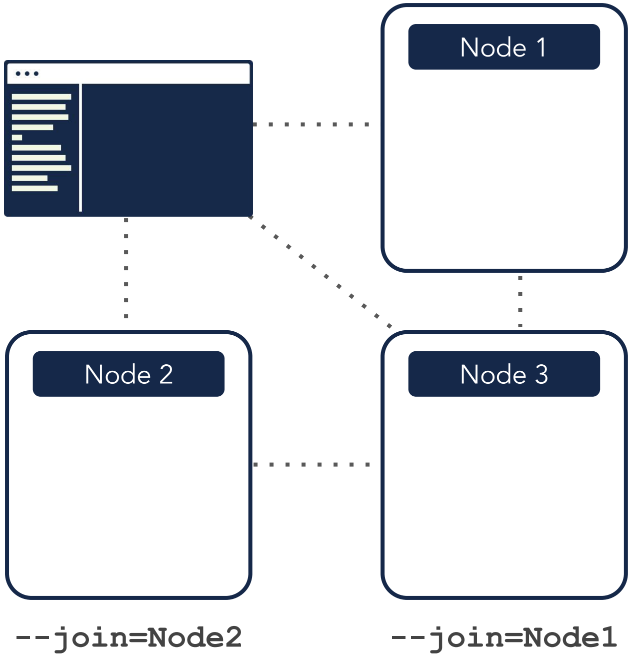
3. **Leaseholder**: Each range designates one one "leaseholder" replica that coordinates all read and write requests for the range. This almost always the Raft Leader for the range.

4. **Consensus**: Writes are confirmed against a consensus when a majority of nodes containing replicas of the range acknowledge the write.



Data can be tied to any place (country, cloud, etc.)

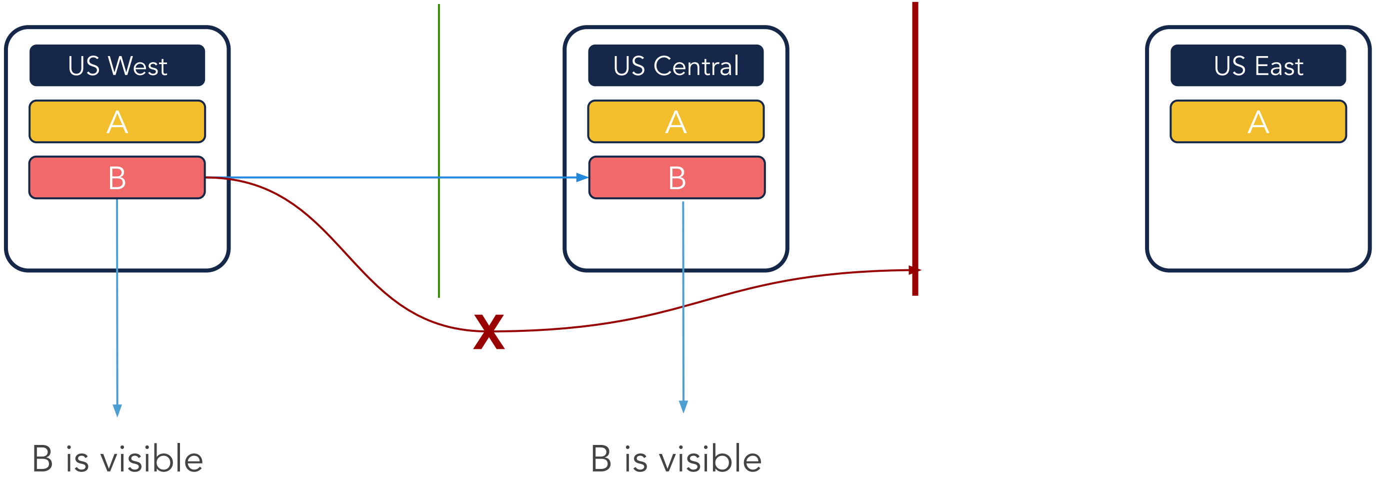
# Simple Deployment with Symmetric Nodes



- Download single binary
- Install with a single command
- Every node is a client gateway
- Each node is a share-nothing member of the cluster.
- Applications see one logical DB

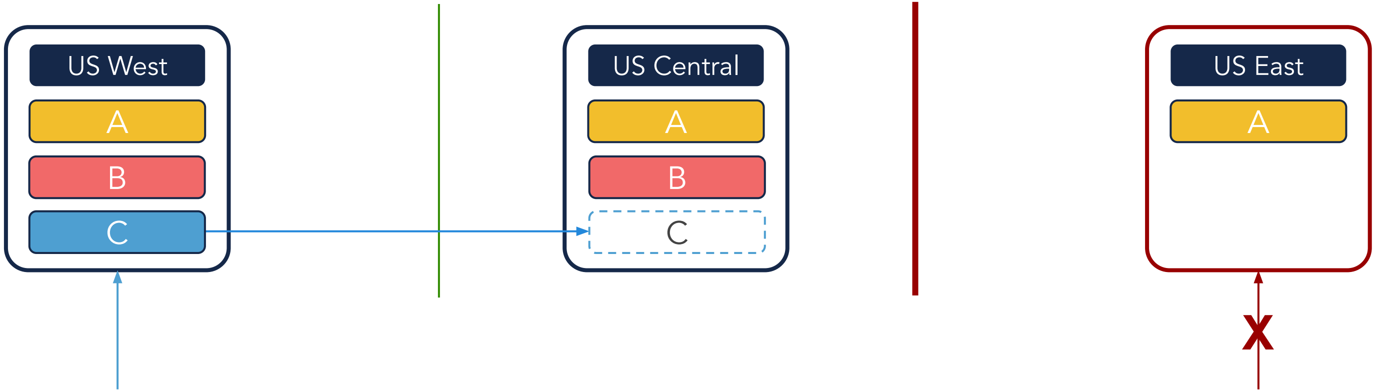
# Consistency And Resiliency Between Datacenters

Raft majority consensus means data is always consistent and available.



# Consistency Between Datacenters

Majority consensus can also write even if some nodes are down.



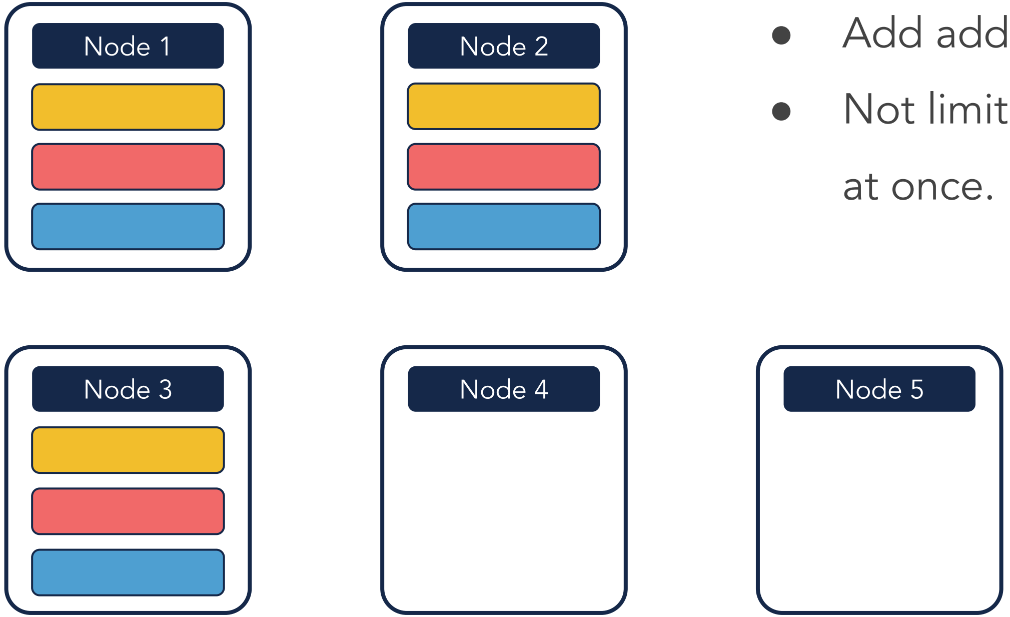
C is writable

C is not writable

Cockroach DB

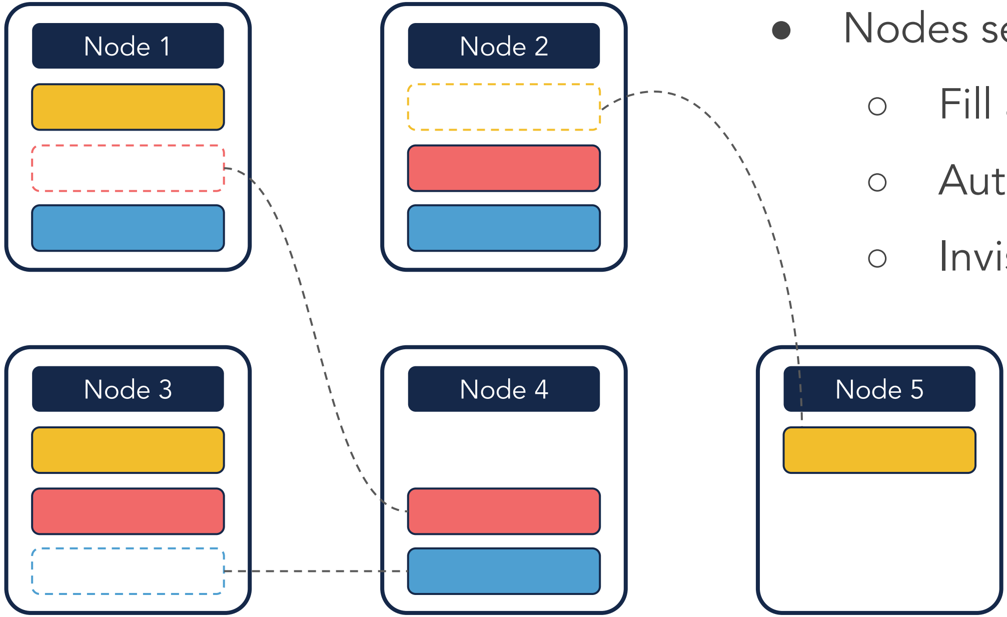
@cockroachdb

# Scale Out



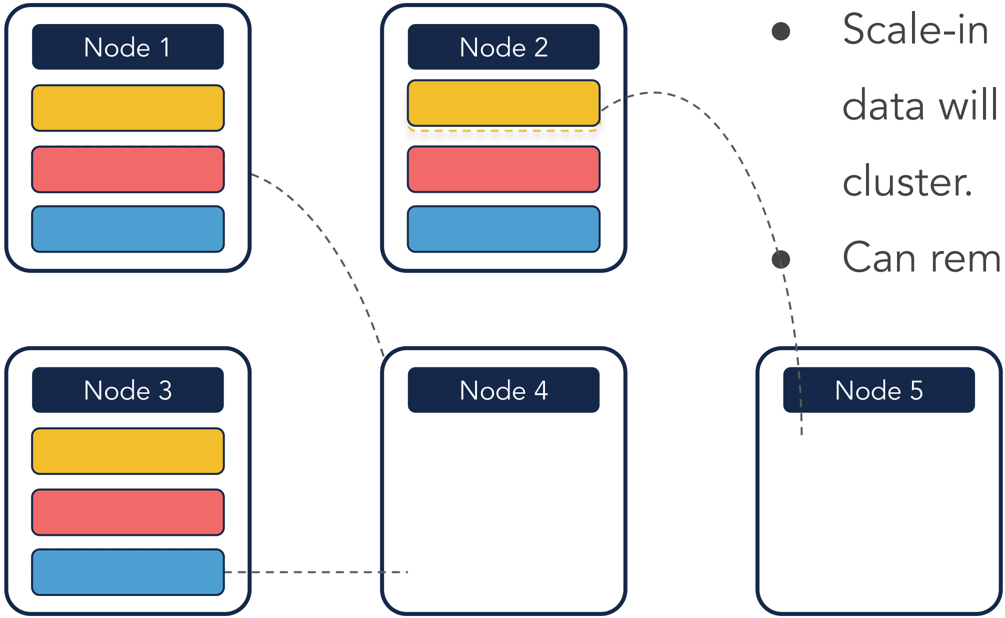
- Add additional nodes to scale
- Not limited to one-at-a-time, can add many at once.

# Scale Out



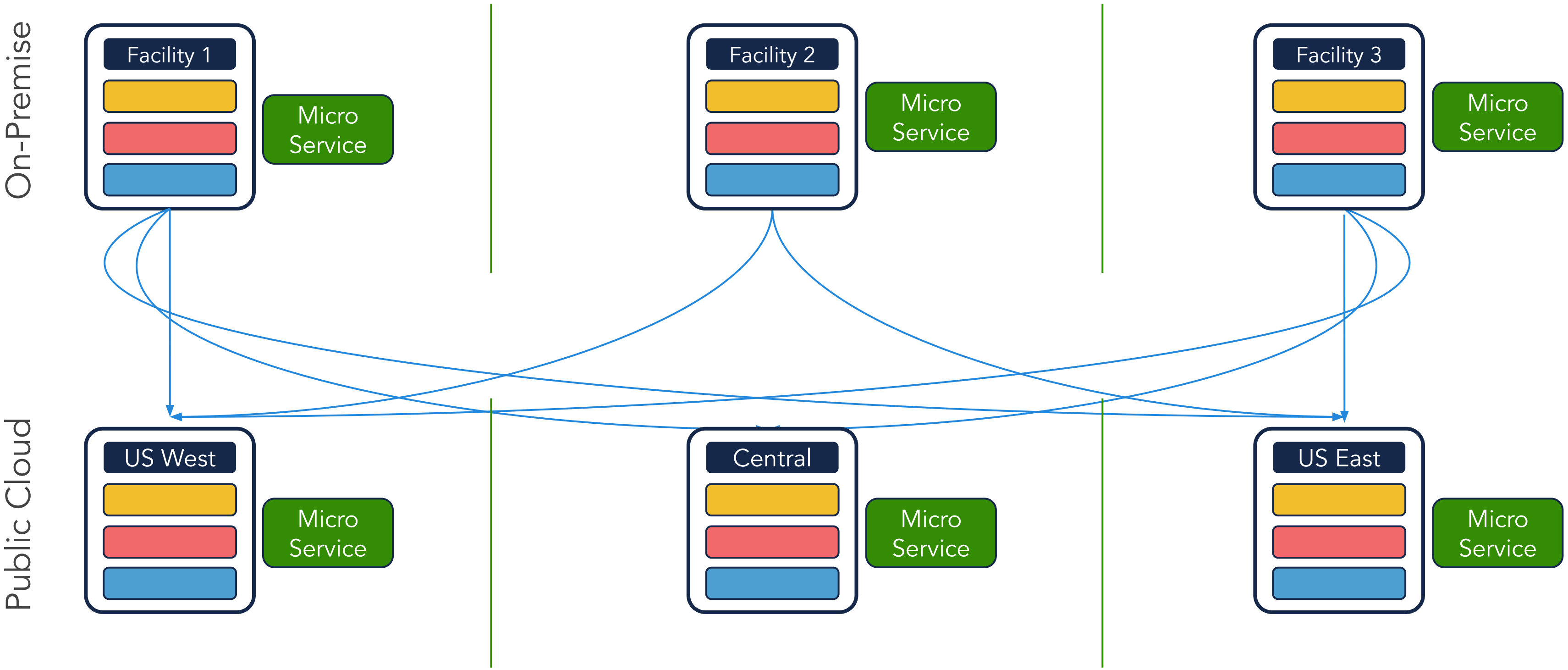
- Nodes self-organize through rebalancing
    - Fill available space from new nodes
    - Automated failover and recovery
    - Invisible to applications

# Scale In



- Scale-in by simply shutting down nodes and data will be moved to the rest of the cluster.
- Can remove many nodes at once.

# Deploy and Migrate Across Clouds



Cockroach DB

@cockroachdb

# Underneath the Hood

Photo: stshank

**Multi-version concurrency control (MVCC)**

- Values are never updated “in place”, newer versions shadow older versions.
- Tombstones are used to delete values.
- Provides snapshot to each transaction.

# Underneath the Hood

Photo: Taki Steve

## Internal Architecture

- Data is stored on disk in a Key-Value database (RocksDB).
- Keys and values are strings lexicographically ordered by key.
- Monolithic key-space.

# Underneath the Hood

**CockroachDB has access to powerful KV primitives**

```
Get(key)

Put(key, value)

ConditionalPut(key, value,
expValue)

Scan(startKey, endKey)

Del(key)
```

## KV Store details

- Natural separation for distributed transactions.
- Atomic columns enable dynamic schema change.
- Extendable to additional functionality such as geo-partitioning.
- CRDB retains the efficiency of a KV store but gains a natural ability to distribute data and still remain SQL.

# Underneath the Hood

| key | value |
|---|---|
| dog/34/name | carl |
| dog/34/weight | 10.1 |
| dog/7A/name | dagne |
| dog/7A/weight | 13.4 |
| dog/94/name | figment |
| dog/94/weight | 65.8 |
| dog/BC/name | jack |
| dog/BC/weight | 49.7 |

## KV Store details

- All tabular data is stored as monolithic sorted map of KV pairs
- All tables have a primary key
- One key/value pair per column and  keys and values are strings
    - Key: <table>/<index>/<key>/<columnName>
    - Value: <columnValue>

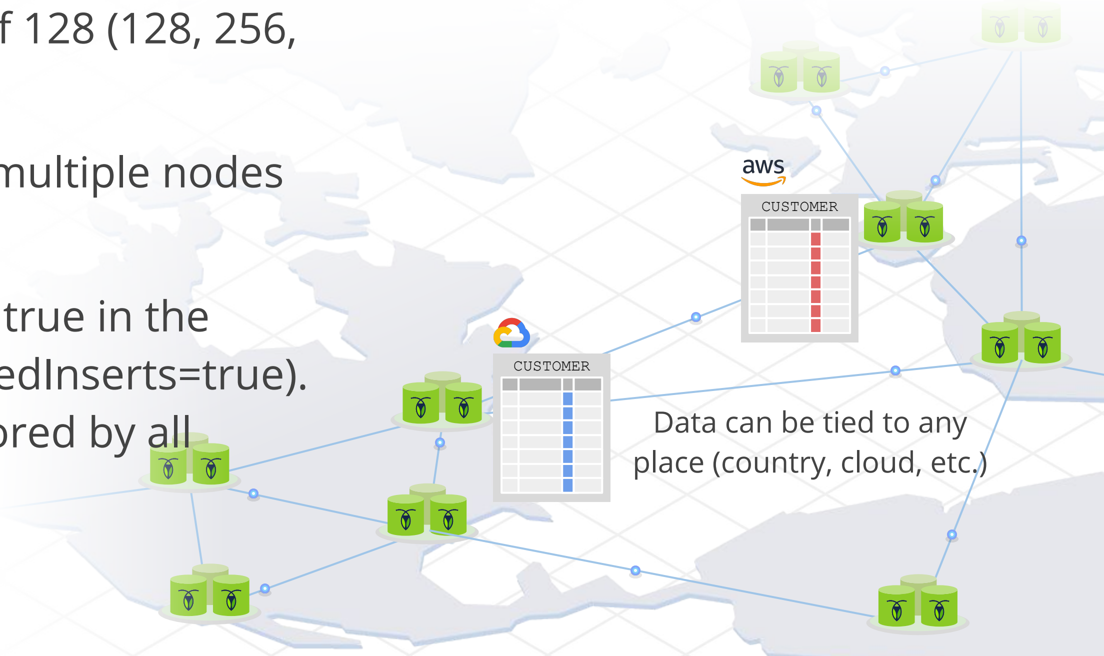
# Optimizing performance in Distributed SQL Databases: Bulk Inserts

1. Use the Import statement for best performance.

2. If you must use Insert, use multi-value inserts with batch sizes in increments of 128 (128, 256, 512, etc).

3. Spread your connection across multiple nodes for higher parallelism.

4. Set the Batched Inserts parm to true in the connection string (reWriteBatchedInserts=true). (But note this is not always honored by all frameworks and drivers.)

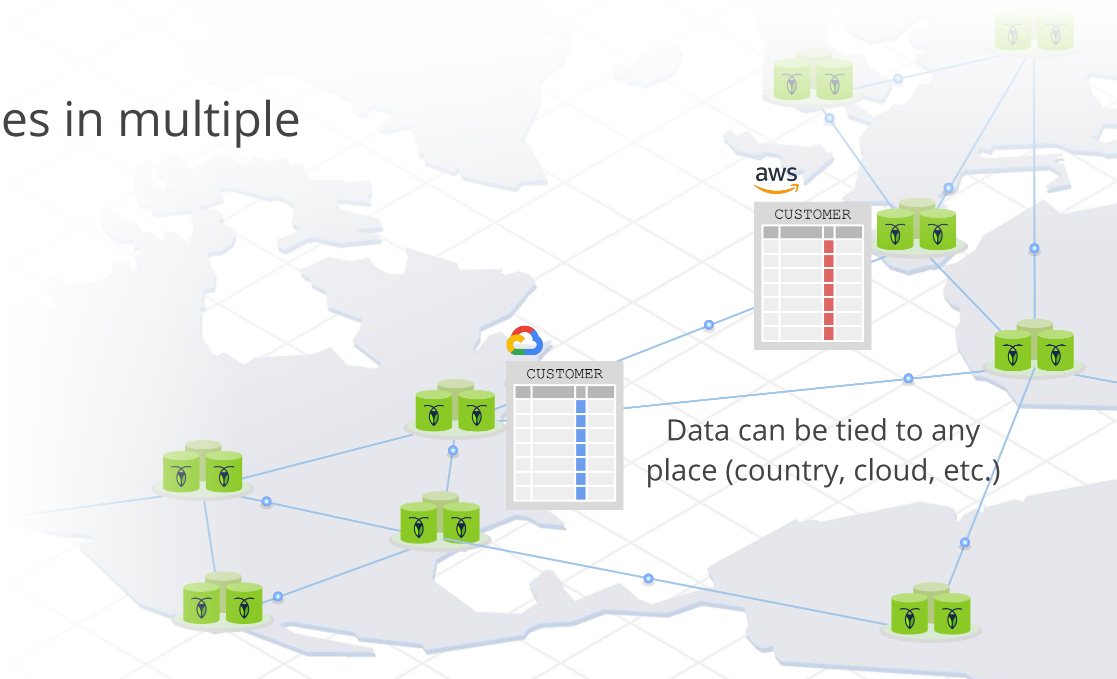


Data can be tied to any place (country, cloud, etc.)

Cockroach LABS

# Optimizing performance in Distributed SQL Databases: Reads

1. **Leaseholder pinning**: You can pin the leaseholders to specified nodes for faster performance.
2. **Local indexes**: Set up indexes in multiple zones for faster reads.



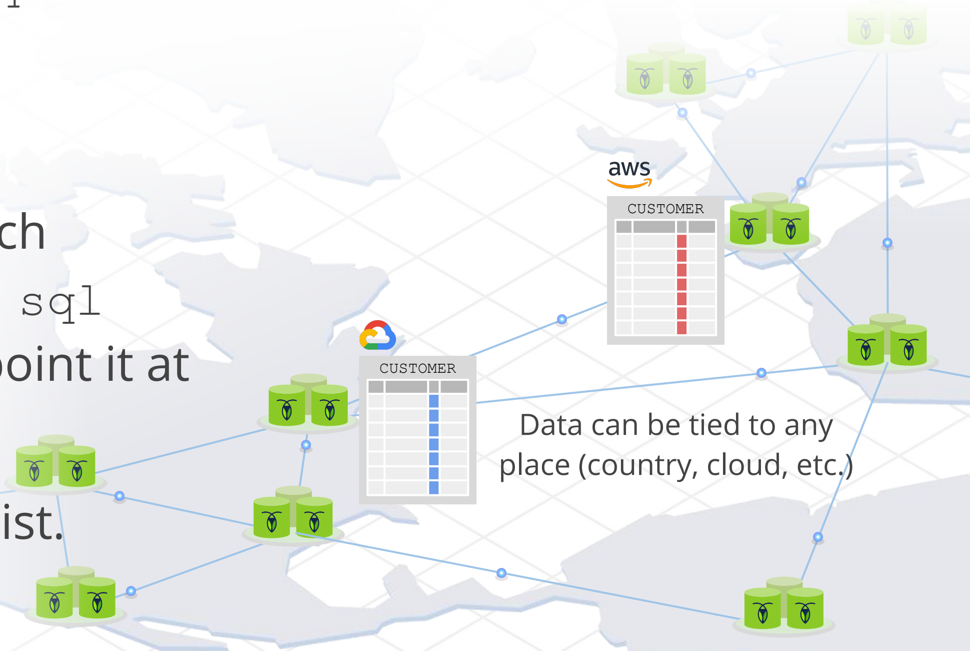
Data can be tied to any place (country, cloud, etc.)

Cockroach LABS

# Connecting to CockroachDB

Connection options for CockroachDB

1. With the SQL shell. (`cockroach sql` from the command line.)
2. With jdbc (jdbc.postgresql.org)
3. Within Kubernetes. (Use a cockroach container and run the `cockroach sql` command and set the `--host` to point it at the internal vip.)
4. See **bit.ly/crdb-connect** for a full list.



Data can be tied to any place (country, cloud, etc.)

# Q & A

Cockroach LABS