yugabyteDB

# End-to-End Stack

## Harnessing YugabyteDB and Ecosystem Solutions to Build Globally Scalable Apps

# Yogi Rampuria

## Principal Solution Engineer, Yugabyte

Channels & Alliances - Asia South & India

@yogendra | yogendra.me | github.com/yogendra

#Singapore #OSSEnthusiast #CoderSince90s #K8s #Java #DevEx

# Why are we here?

Market Trends / On the Rise

- Cloud Native Applications
- Cloud Adoption
- Multi-Cloud and Multi-Region Architectures

CNA + Container + Kubernetes + Multi-Cloud = 🤯

“Will it work?”🙏
“Will this work with that?”🤔
“Why is it not working?”😡

## “Why is it working?”🤷‍♂️

So,

Let’s go to the beginning..

# 14 Billion Years Ago…

🤣🤣🤣

Kidding

## It really starts with..

An App

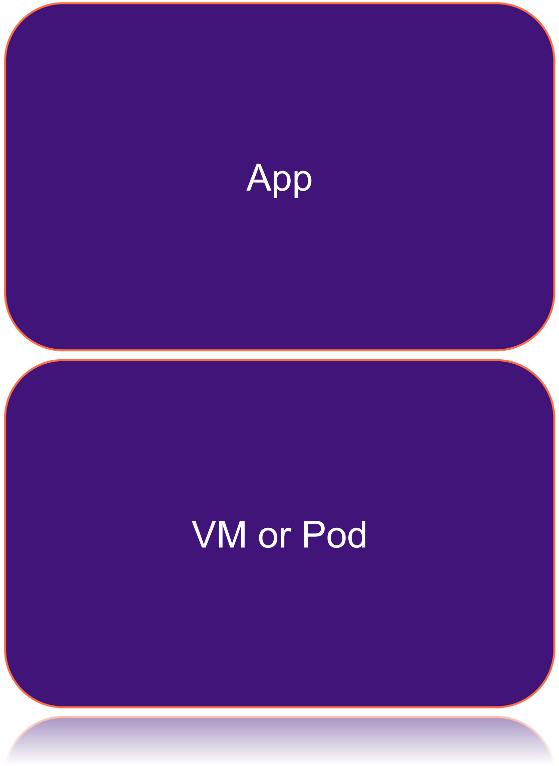
# What is an Application/App?

App on Computer/Mobile?
Software?
A piece of Code?

# What is an Application/App?

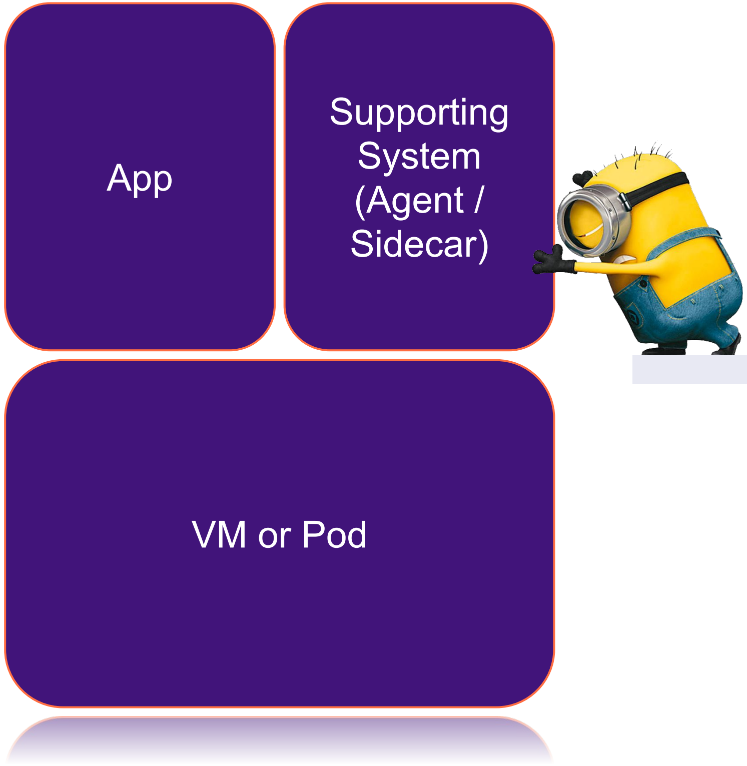
~~App on Computer/Mobile?~~
~~Software?~~
A piece of Code?

# How do you run it?


App

VM or Pod

# And there is a bit More…

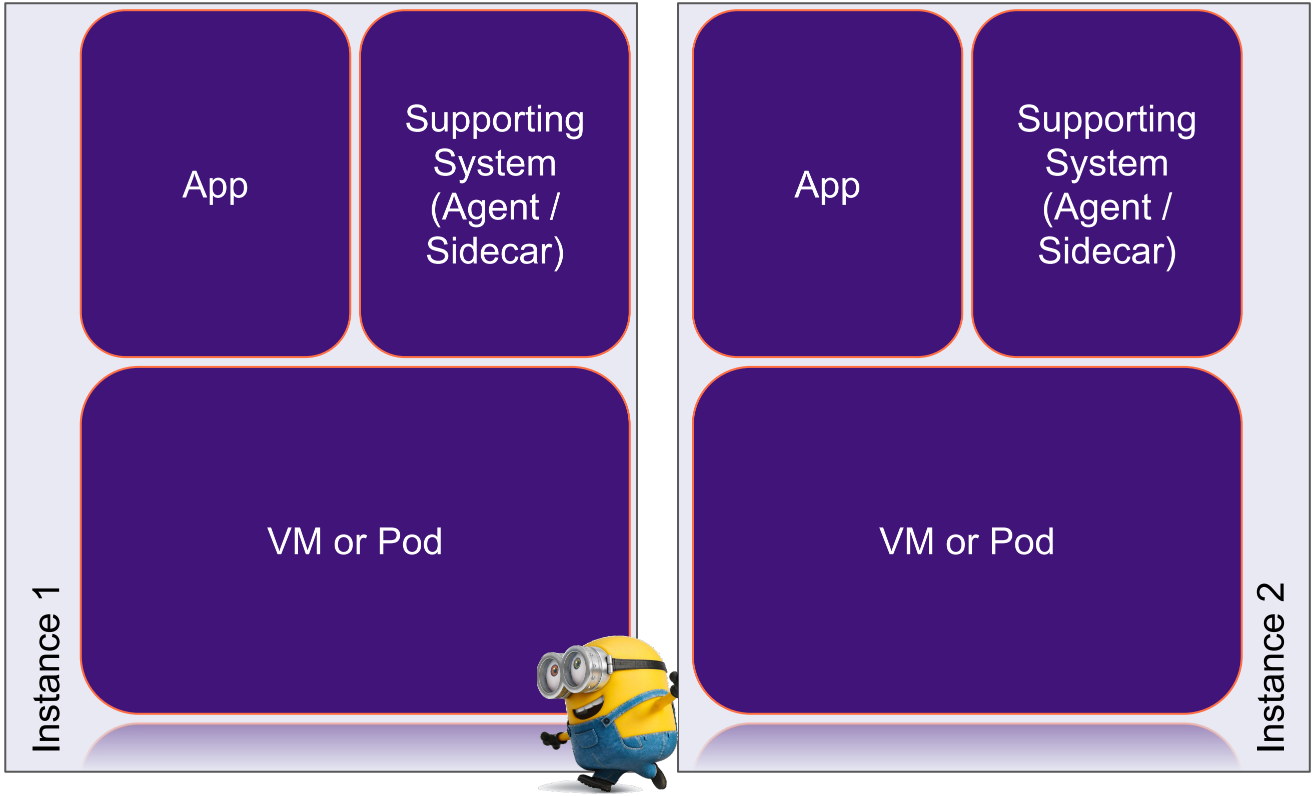


App

Supporting System (Agent / Sidecar)

VM or Pod

# And never alone



App

Supporting System (Agent / Sidecar)

VM or Pod

Instance 1

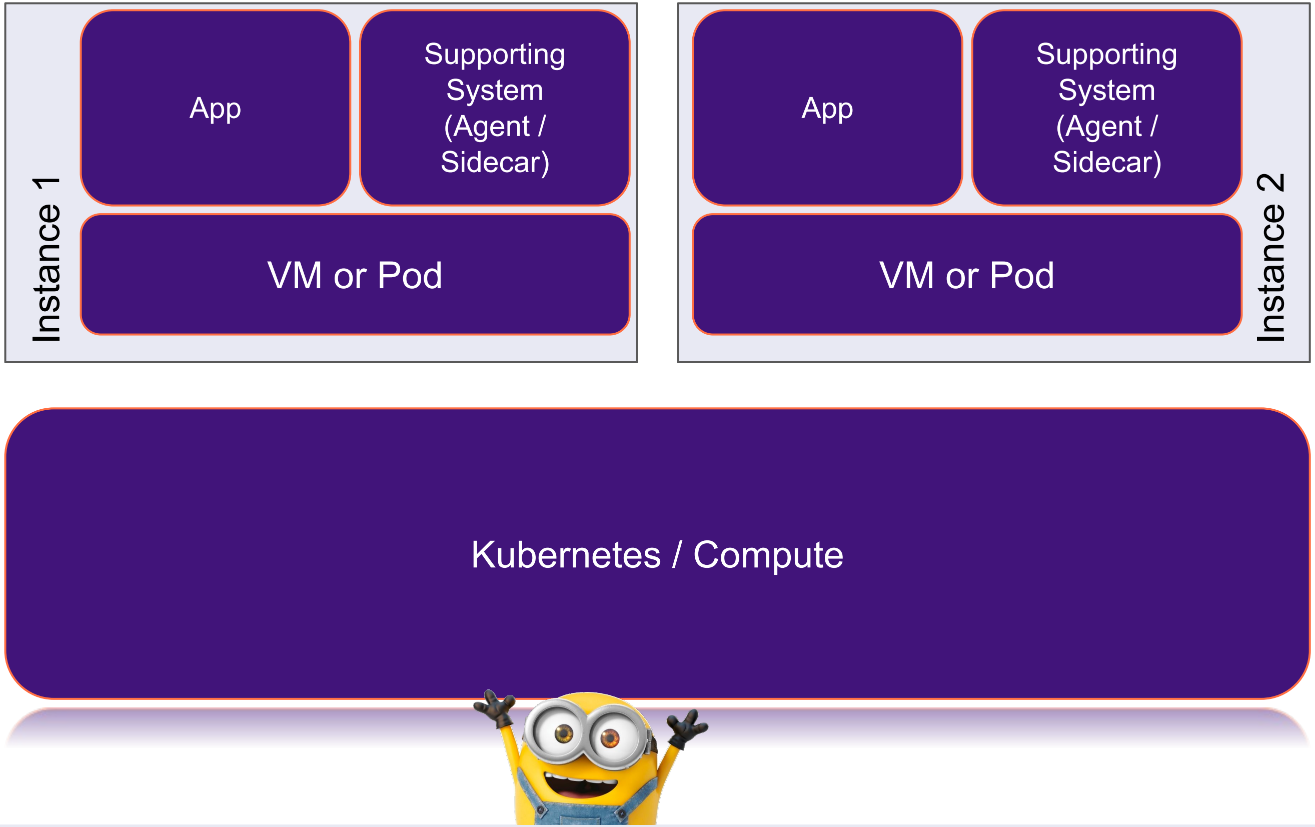
App

Supporting System (Agent / Sidecar)

VM or Pod

Instance 2

# And it runs on a platform



Instance 1

App

Supporting System (Agent / Sidecar)

VM or Pod

Instance 2

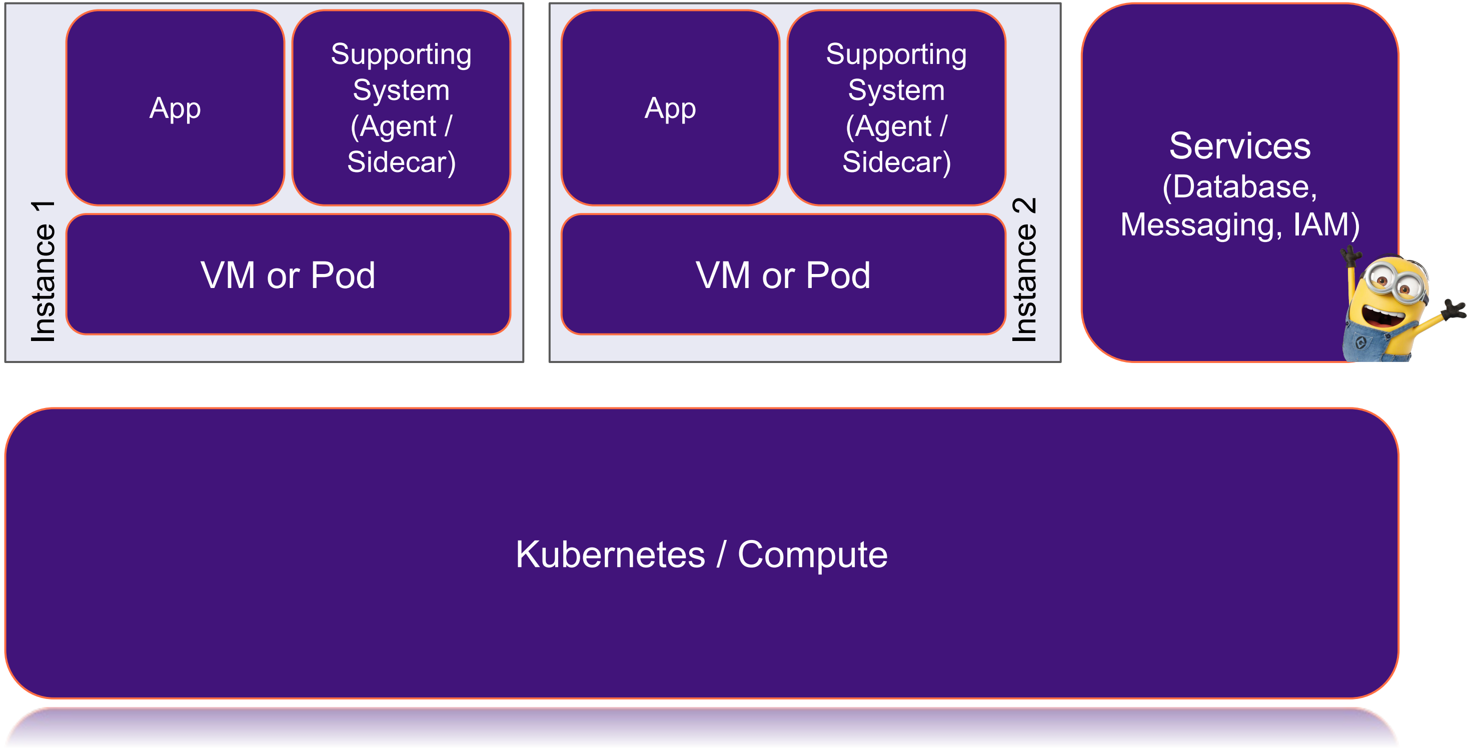
App

Supporting System (Agent / Sidecar)

VM or Pod

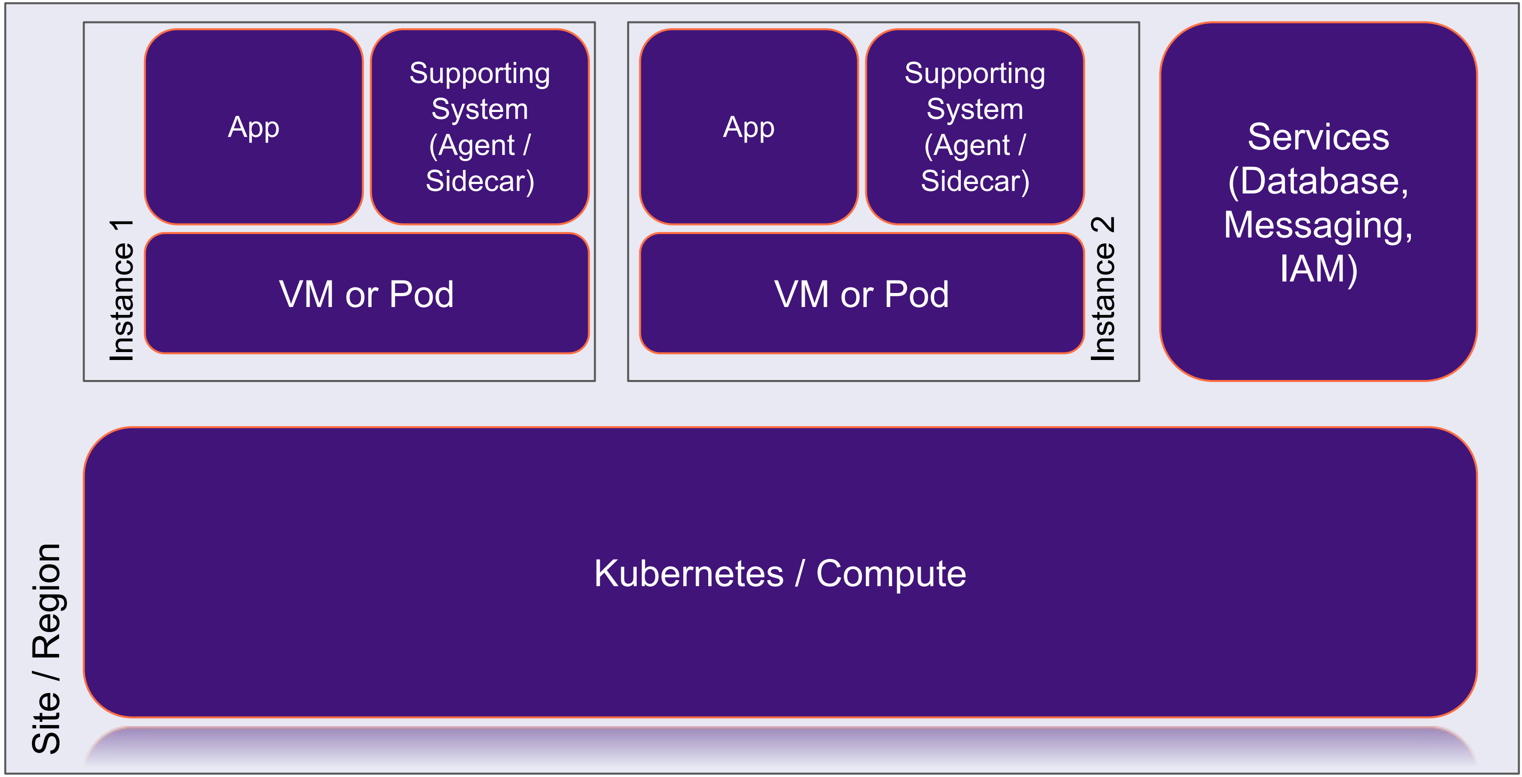
Kubernetes / Compute

# With some supporting services

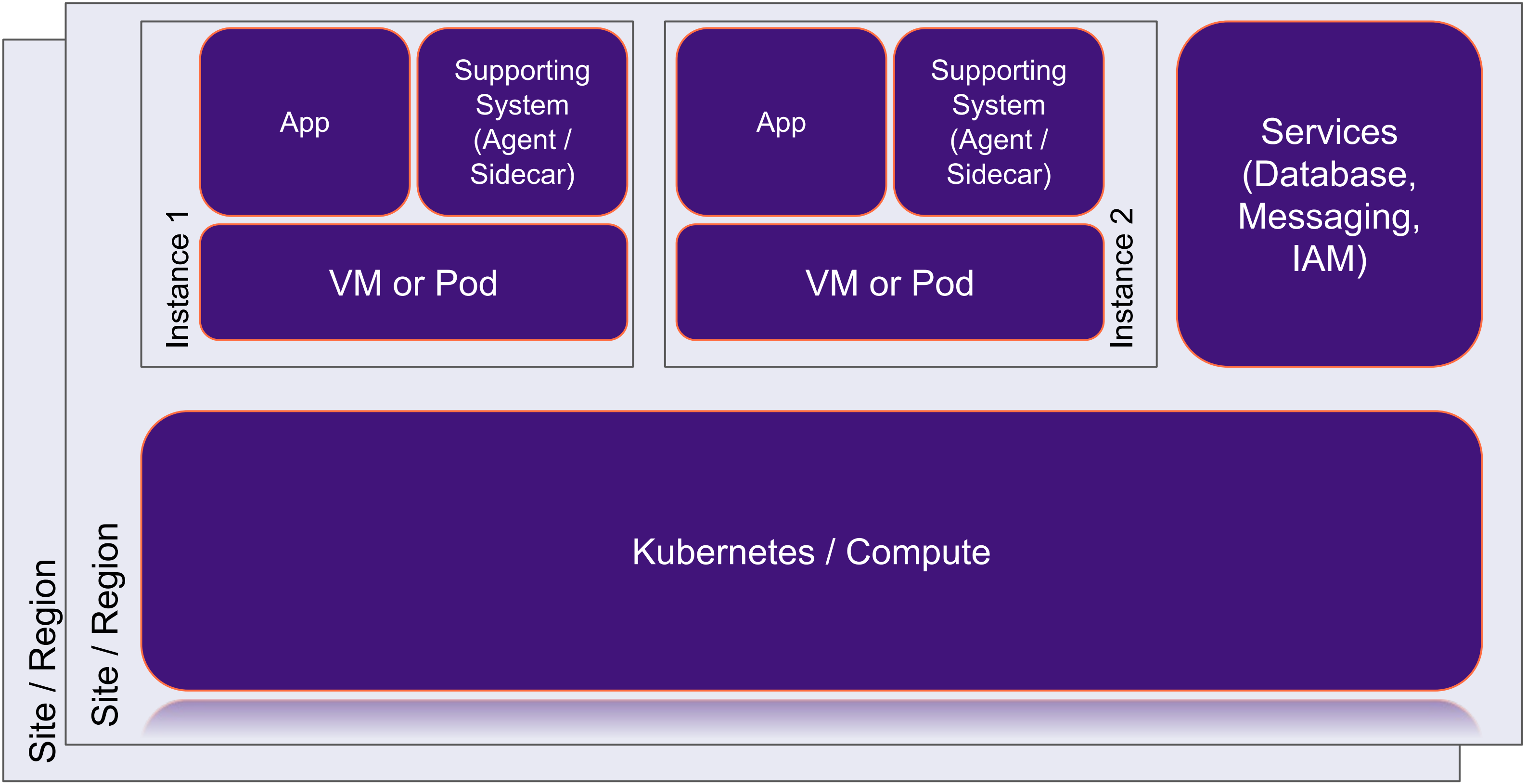


Instance 1
App
Supporting System (Agent / Sidecar)
VM or Pod
App
Supporting System (Agent / Sidecar)
VM or Pod
Instance 2
Services (Database, Messaging, IAM)
Kubernetes / Compute

# Lets not forget Site


App
Supporting System (Agent / Sidecar)
Instance 1
VM or Pod
App
Supporting System (Agent / Sidecar)
Instance 2
VM or Pod
Services (Database, Messaging, IAM)
Site / Region
Kubernetes / Compute

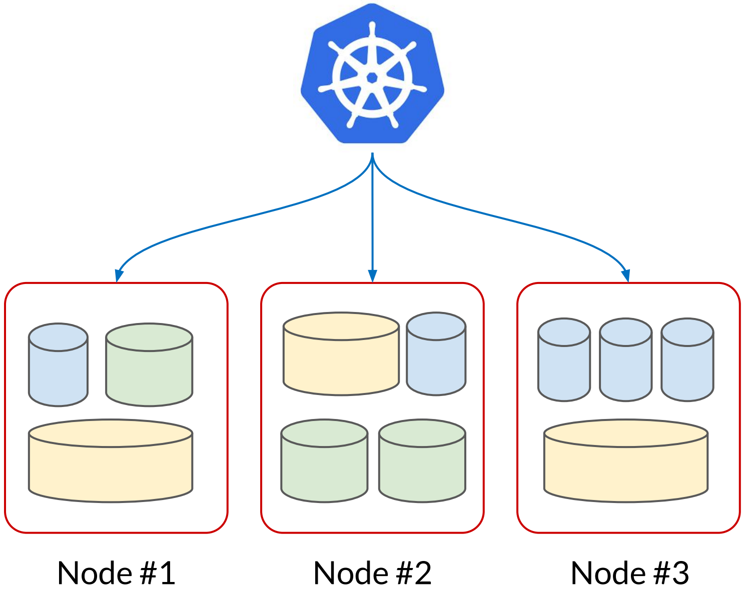
# And there is more than one


App
Supporting System (Agent / Sidecar)
VM or Pod
Instance 1
App
Supporting System (Agent / Sidecar)
VM or Pod
Instance 2
Services (Database, Messaging, IAM)
Kubernetes / Compute
Site / Region
Site / Region

# Why run a DB in K8s?

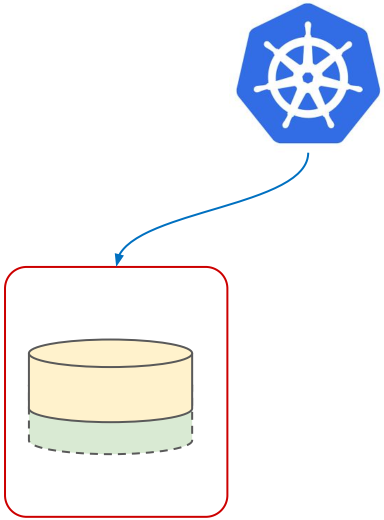
# Better resource utilization

- Reduce cost with better packing of DBs
- Useful when running large number of DBs
  - Multi-tenant applications with a DB per tenant
  - Self-service private **DBaaS**
- But watch out for noisy neighbors
  - Perf issues when running critical production workloads



# Resize pod resources dynamically

- Dynamically change CPU, memory
- Embrace Automation - done without incurring downtime
  - Scale DB with workload
  - Automate to scale up automatically

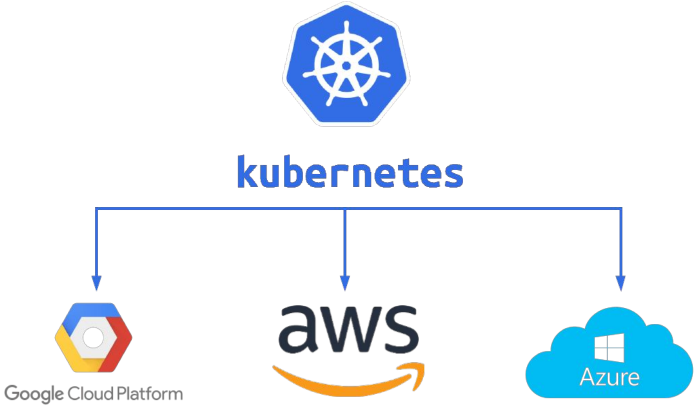


```
$ kubectl apply -f cpu-request-limit.yaml

$ kubectl apply -f
memory-request-limit.yaml
```

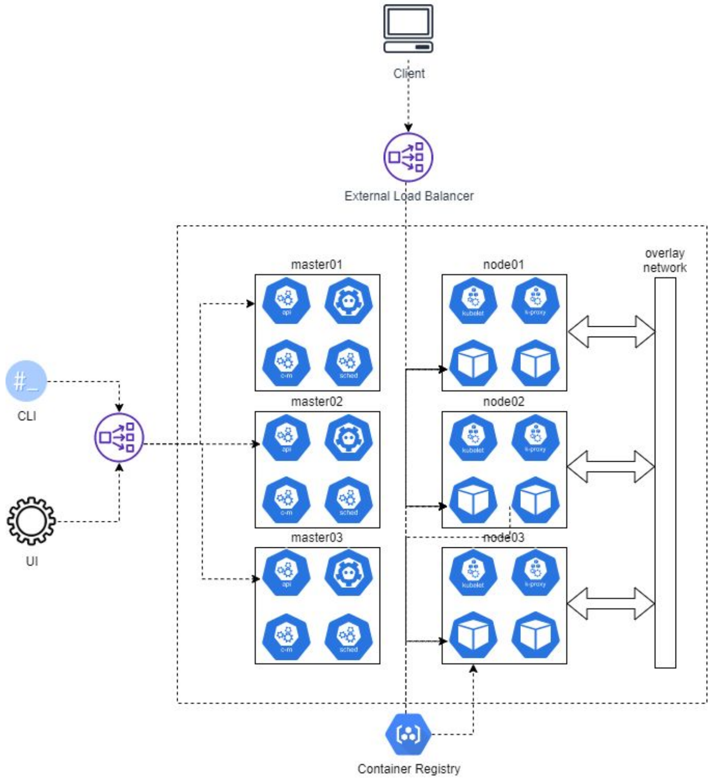
# Portability between clouds and on-premises

- Infrastructure as code
- Works in a similar fashion on any cloud
  - Cloud-provider managed k8s (AKS, EKS, GKE)
  - Self-managed k8s (public/private cloud)
- But not perfectly portable
  - Need to understand some cloud specific constructs (Example: volume types, load balancers)



# Out of box infrastructure orchestration

- Pods that fail are automatically restarted
- Pods are resized across nodes in cluster
  - Optimal resource utilization
  - Specify policies in code (example: anti-affinity)
- Loss of some flexibility
  - Cannot make permanent changes on pods



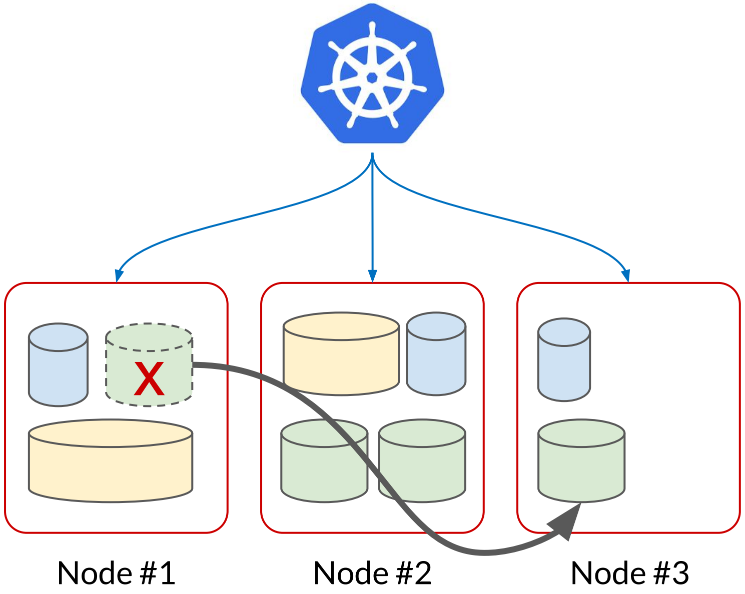
# Automating day 2 operations

- Robust automation with **CRDs** (Custom Resource Definitions) or commonly referred as ‘K8s Operator’
- Easy to build an operator for ops
  - Periodic backups
  - DB software upgrades
- Automating failover of traditional RDBMS can be dangerous
  - Potential for data loss?
  - Mitigation: use a distributed DB

# Why NOT run a DB in K8s?

# Greater chance of pod failures

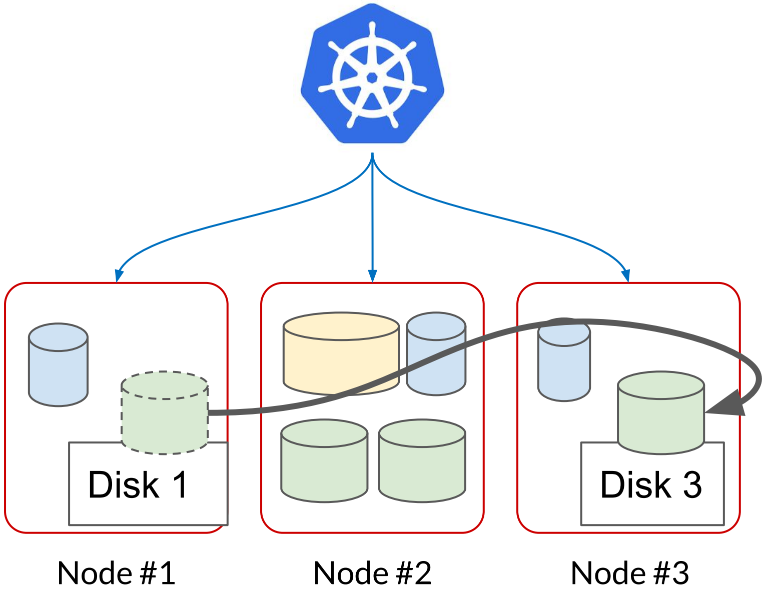
- Pods fail more often than VMs or bare metal
- Many reasons for increased failure rate
  - Process failures - config issues or bugs
  - Out of memory and the *OOM Killer*
  - Transparent rescheduling of pods
- Will pod failures cause disruption of the service or data loss?
  - Mitigation: use a distributed DB



Data loss likely if local storage used by pod

# Local vs persistent storage

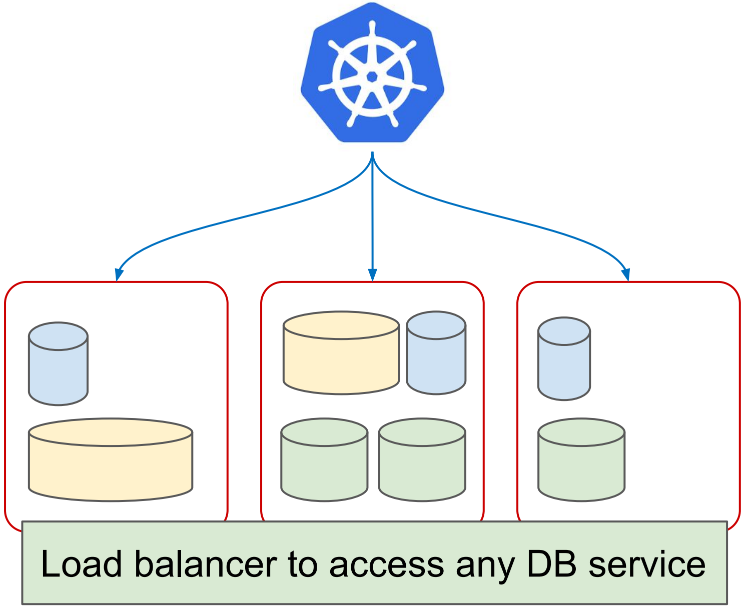
- Local storage = use local disk on the node
  - Not replicated, but higher performance
  - Data not present in new pod location
- Persistent storage = use replicated storage
  - Data visible to pod after it moves to new node
  - What to do for on-prem? Use software solution (additional complexity)
- Mitigation: use a distributed DB



Pod sees a new, empty disk (Disk 3) after move with local storage

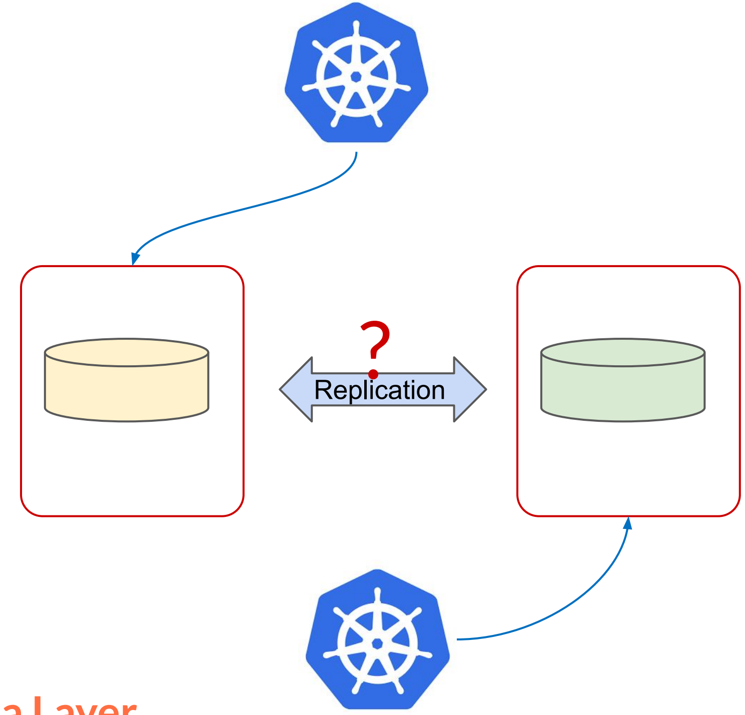
# Need for a load balancer

- Restricted cluster ingress in k8s
    - If app not on same k8s cluster, needs LB
- Needs load balancer to expose DB externally
    - Not an issue on public clouds - use cloud-provider network LBs
    - But there may be per-cloud limits on NLBs and public IP address limits
- Bigger problem on-prem with hardware based load balancers (Example: F5)



# Networking complexities

- Two k8s clusters cannot “see” each other
- Network discovery and reachability issues
  - Pods of one k8s cluster cannot refer and replicate to pods in another k8s cluster by default
- Mitigation #1: use DNS chaining today (operational complexity, depends on env)
- Mitigation #2: use service mesh like Istio (but lower performance - HTTP layer vs TCP)



[Video: Kubecon EU 2021 - Building the Multi-Cluster Data Layer]

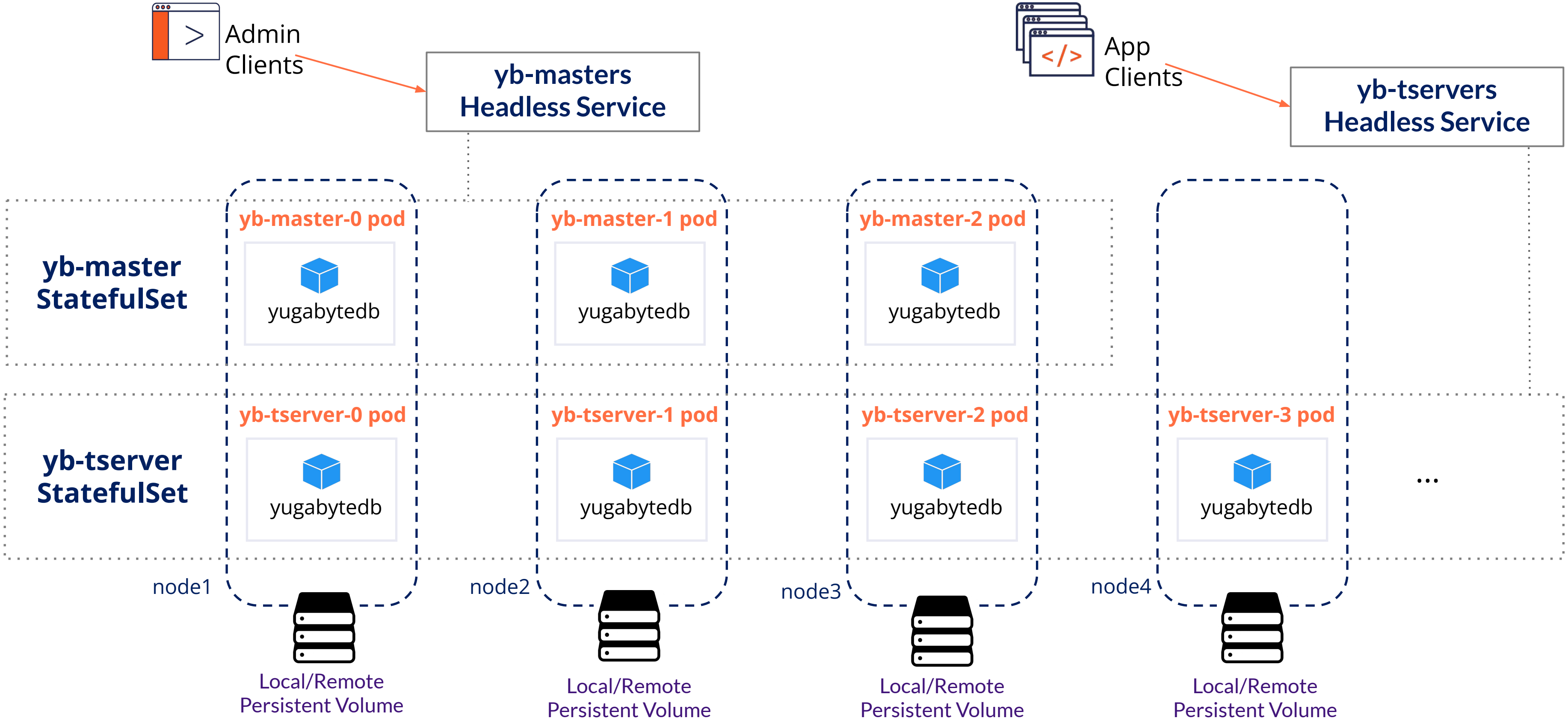
# Run Distributed SQL on Kubernete

# Running a Distributed SQL DB in k8s (YugabyteDB)

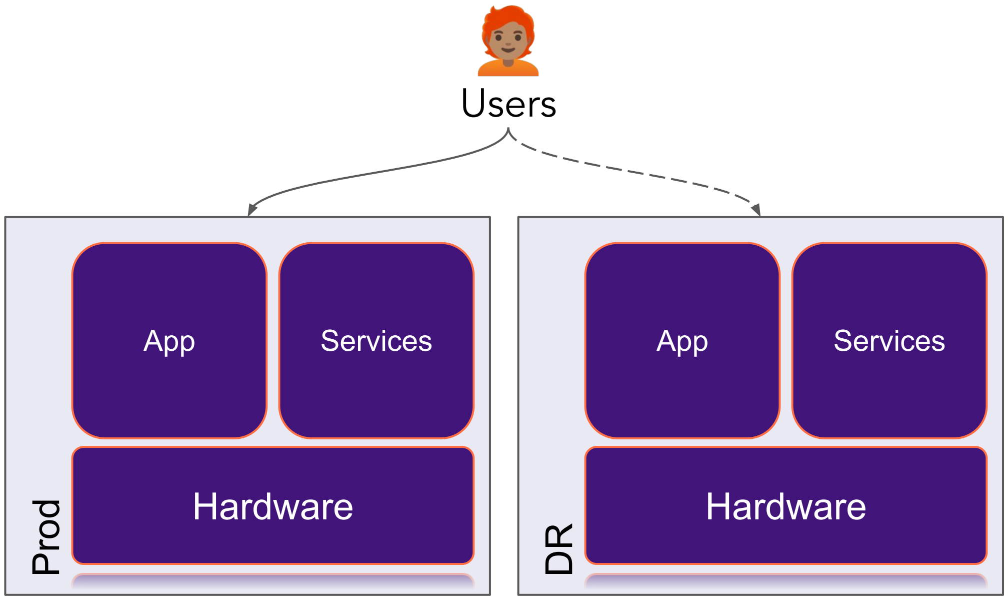
- Better resource utilization
- Resize pod resources dynamically
- Portability (cloud and on-premises)
- Out of box infrastructure orchestration
- Automate day 2 DB operations

- ~~Greater chance of pod failures~~
- ~~Local storage vs persistent storage~~
- Need for a load balancer
- Networking complexities
- Operational maturity curve

# YugabyteDB on Kubernetes

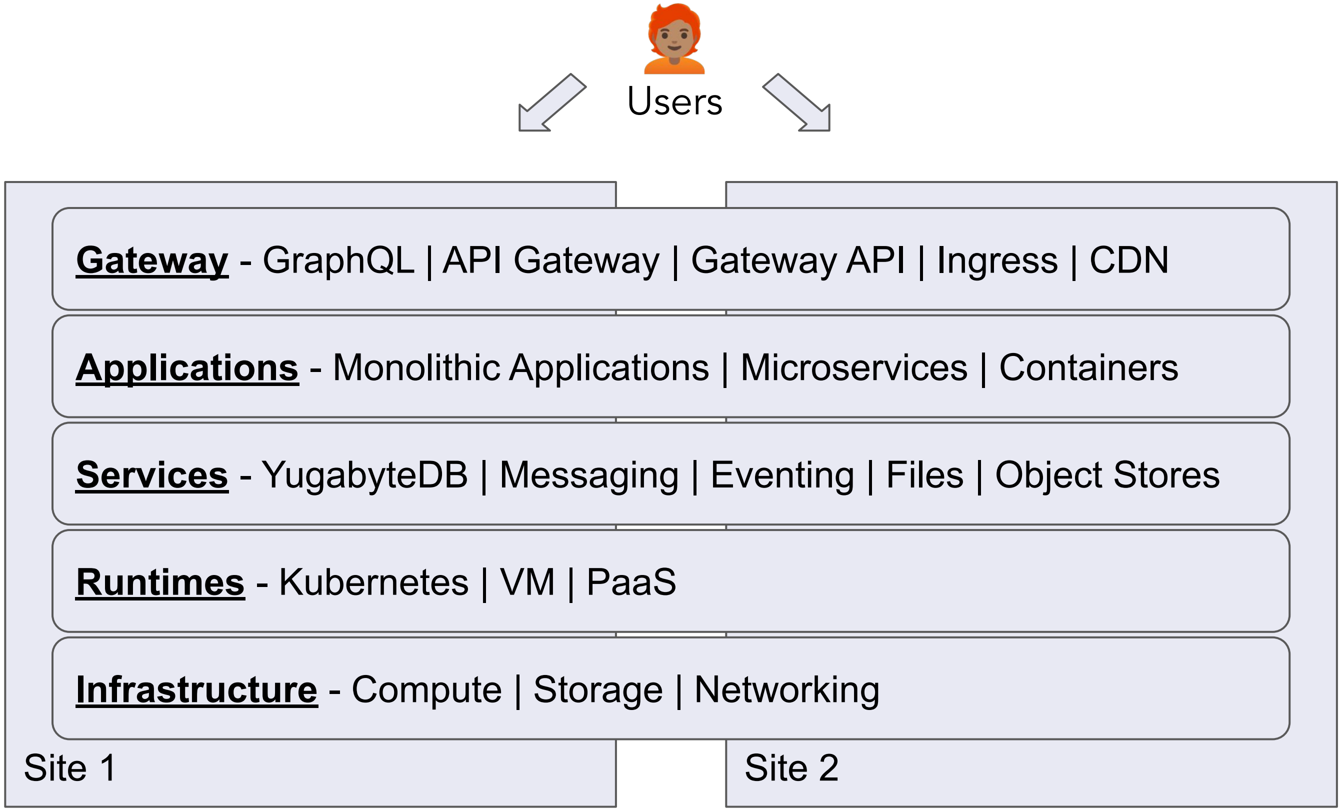


# Common Architecture - Active - Passive / Prod - DR Setup



- All users are served from single site
- Failover - Failback
  - Hours to normalcy
- DNS or GSLB for Application failover
- State/Data replication technology
- Manual actions
  - Error prone
- Systems and Apps are "Pets"

# Cloud Native Architecture: Active - Active



Users

**Gateway** - GraphQL | API Gateway | Gateway API | Ingress | CDN

**Applications** - Monolithic Applications | Microservices | Containers

**Services** - YugabyteDB | Messaging | Eventing | Files | Object Stores

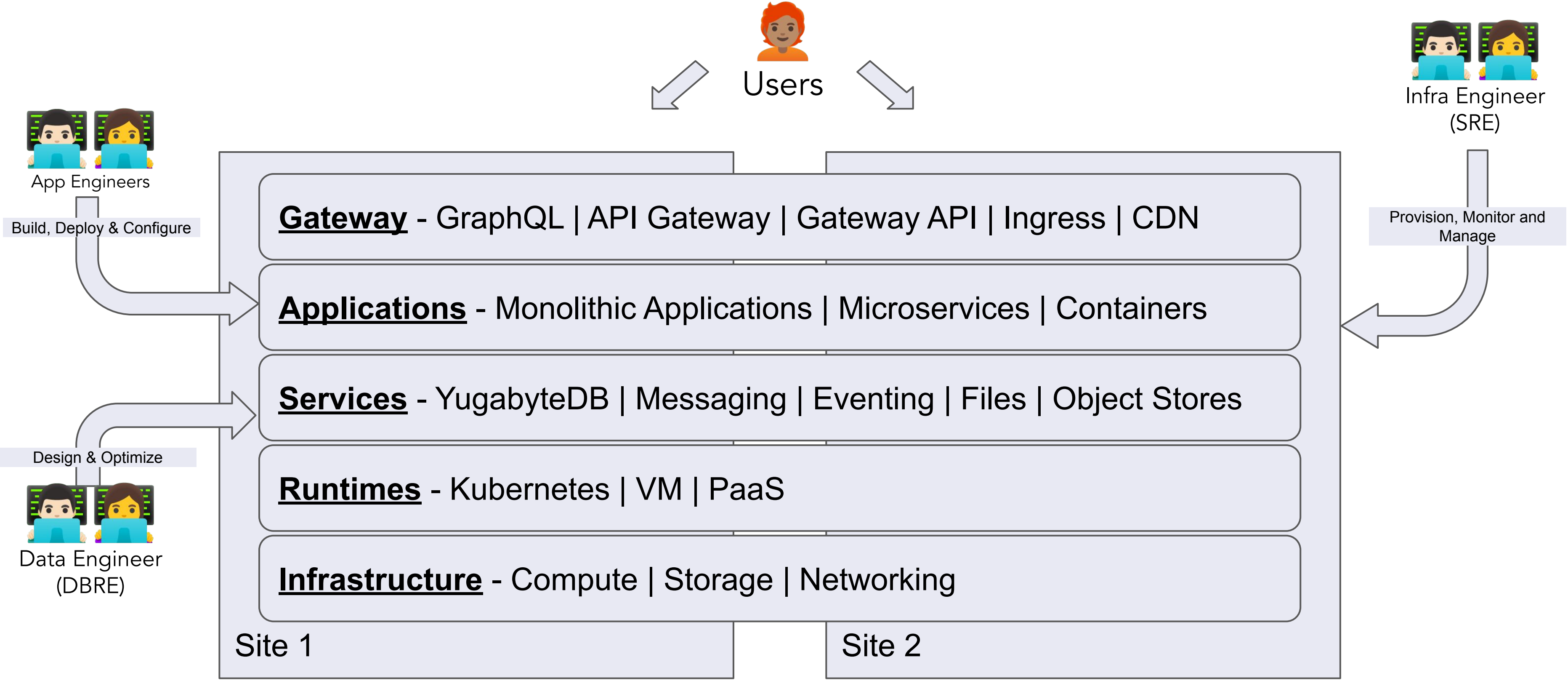
**Runtimes** - Kubernetes | VM | PaaS

**Infrastructure** - Compute | Storage | Networking

Site 1

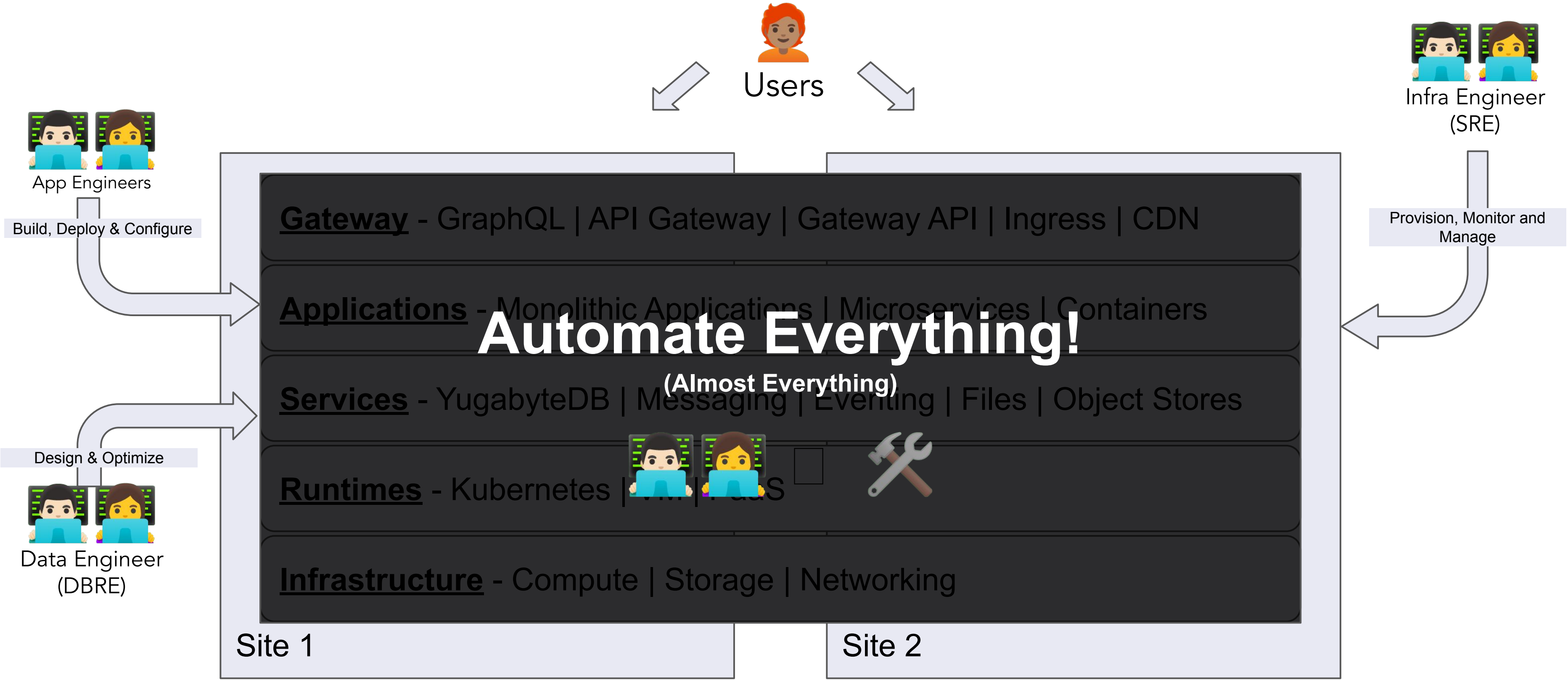
Site 2

# And Evolving Personas



Users
App Engineers
Build, Deploy & Configure
Infra Engineer (SRE)
Provision, Monitor and Manage
Data Engineer (DBRE)
Design & Optimize
**Gateway** - GraphQL | API Gateway | Gateway API | Ingress | CDN
**Applications** - Monolithic Applications | Microservices | Containers
**Services** - YugabyteDB | Messaging | Eventing | Files | Object Stores
**Runtimes** - Kubernetes | VM | PaaS
**Infrastructure** - Compute | Storage | Networking
Site 1
Site 2

# And Evolving Personas



Users

App Engineers

Build, Deploy & Configure

Design & Optimize

Data Engineer (DBRE)

Infra Engineer (SRE)

Provision, Monitor and Manage

**Gateway** - GraphQL | API Gateway | Gateway API | Ingress | CDN

**Applications** - Monolithic Applications | Microservices | Containers

**Services** - YugabyteDB | Messaging | Eventing | Files | Object Stores

**Runtimes** - Kubernetes | VM | PaaS

**Infrastructure** - Compute | Storage | Networking

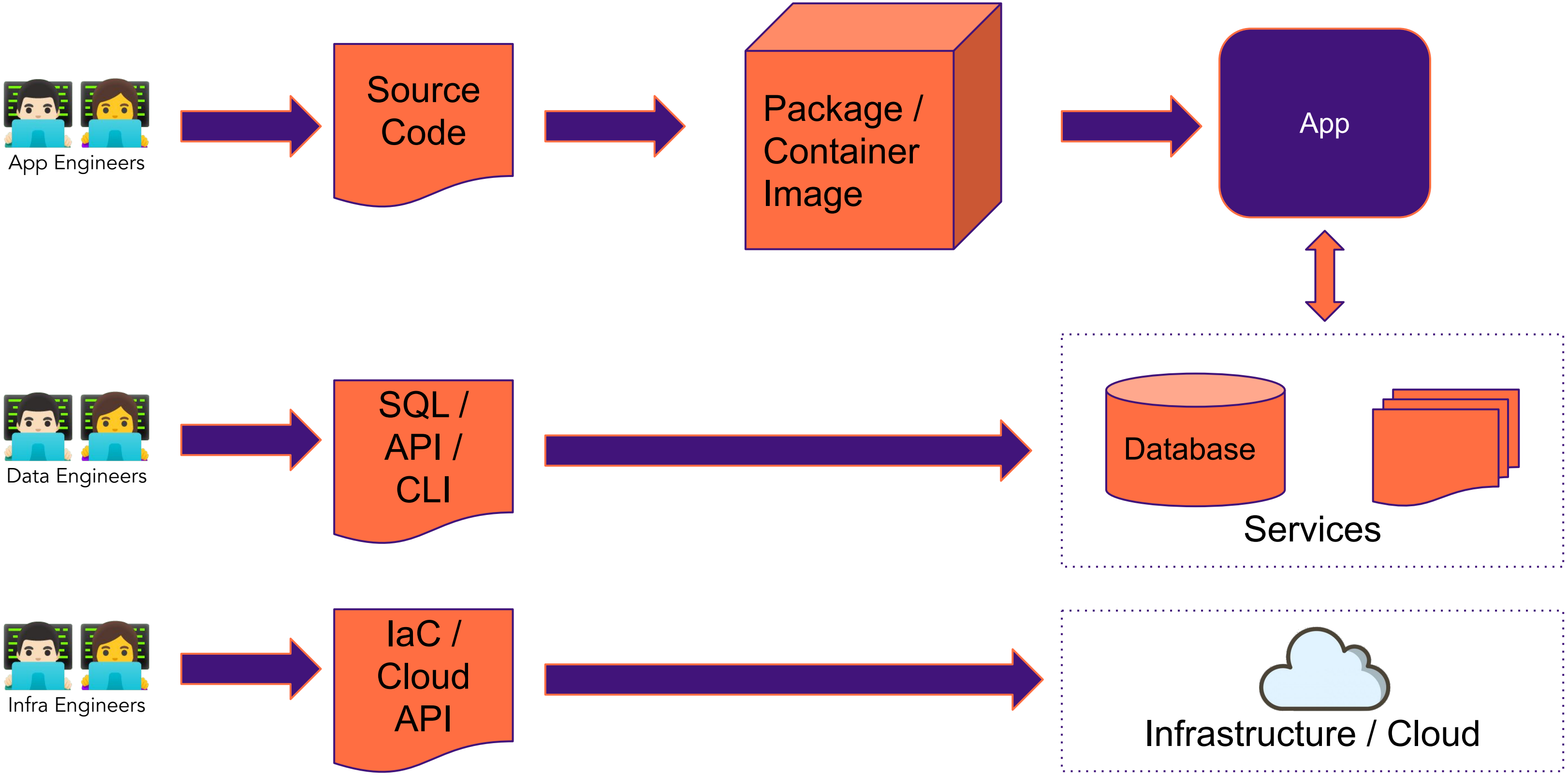
Site 1

Site 2

**Automate Everything!**

**(Almost Everything)**

# CI/CD & DevOps => FTW

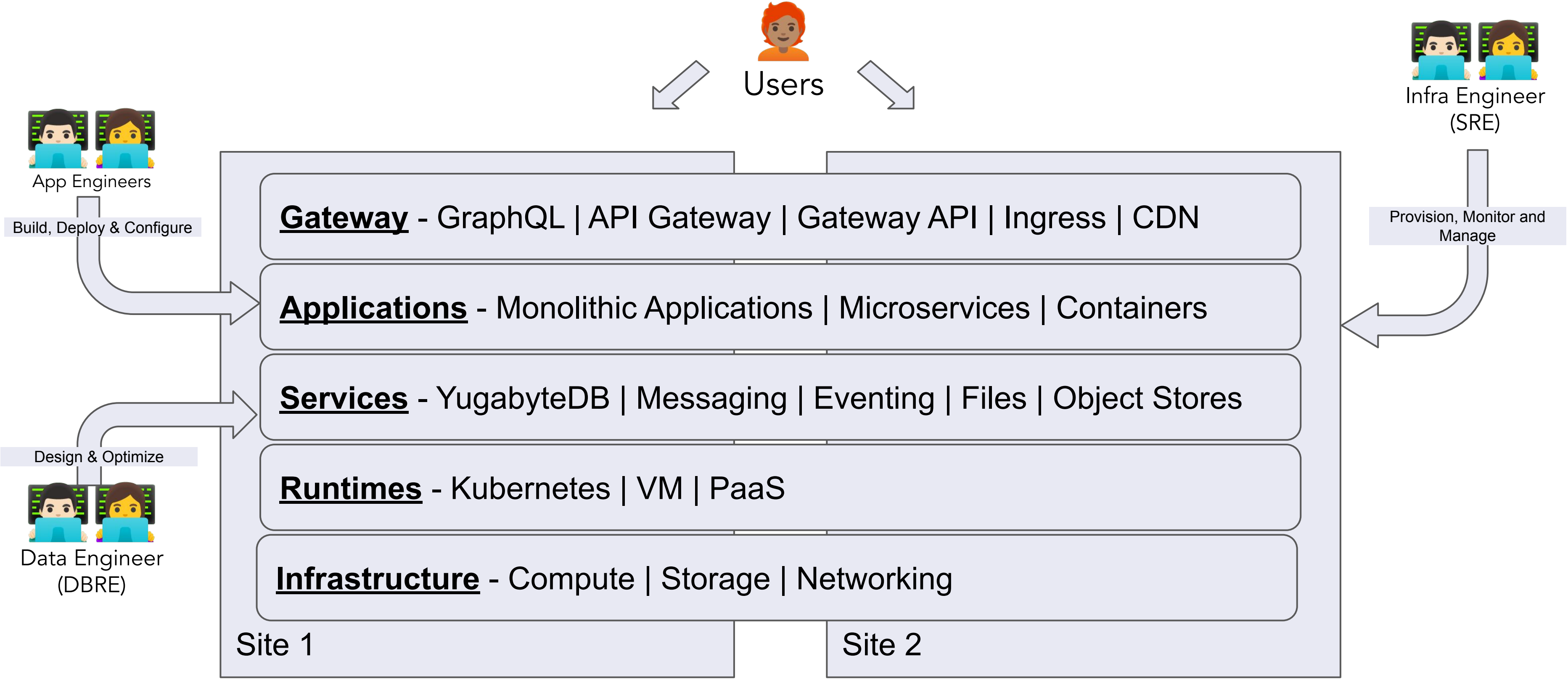


## Demo #1: Inner Loop

# Create an application
# Connect to Database
# E2E testing with Database

**https://github.com/yogendra/yb-e2e-demo/**

# And Evolving Personas



Users

App Engineers

Build, Deploy & Configure

Infra Engineer (SRE)

Provision, Monitor and Manage

Design & Optimize

Data Engineer (DBRE)

**Gateway** - GraphQL | API Gateway | Gateway API | Ingress | CDN

**Applications** - Monolithic Applications | Microservices | Containers

**Services** - YugabyteDB | Messaging | Eventing | Files | Object Stores

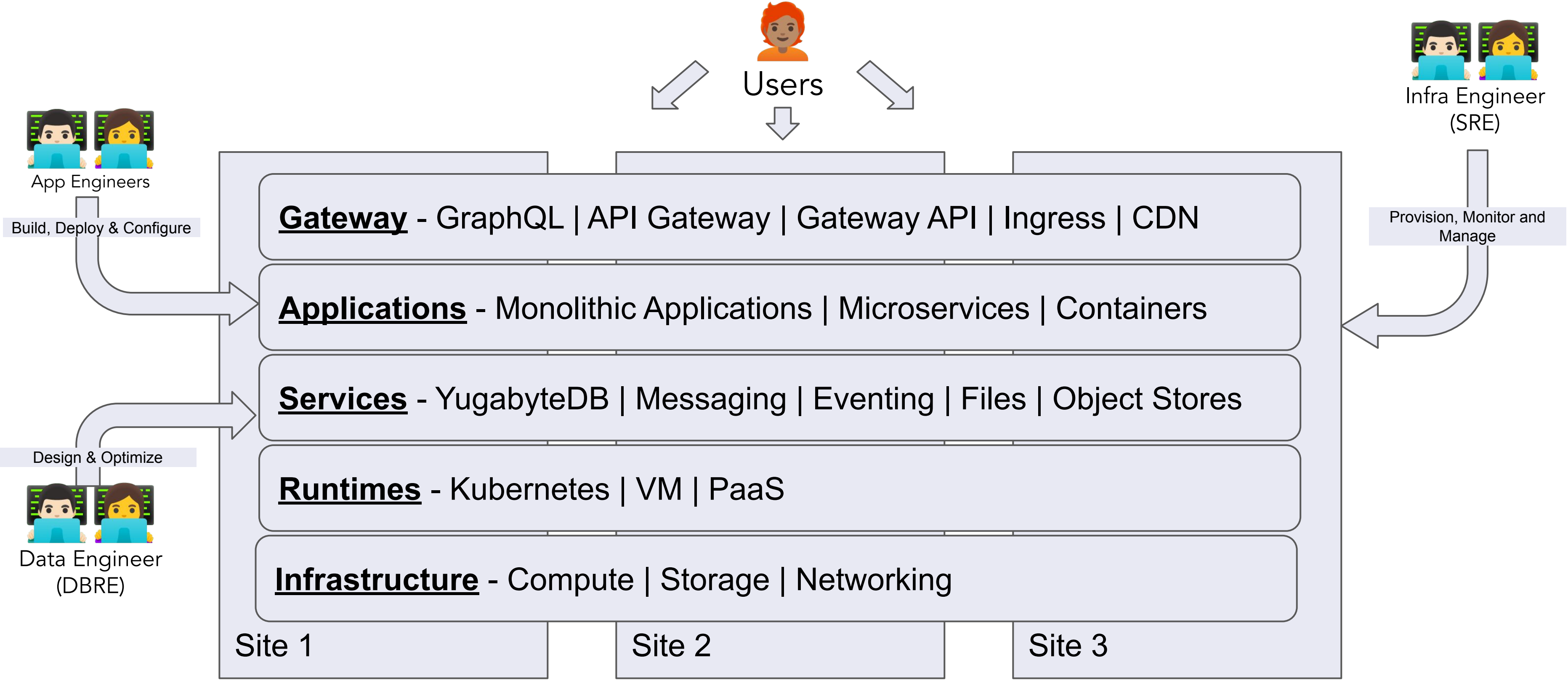
**Runtimes** - Kubernetes | VM | PaaS

**Infrastructure** - Compute | Storage | Networking

Site 1

Site 2

# Let’s Add One More Site - Just for Fun



Users

App Engineers

Build, Deploy & Configure

Data Engineer (DBRE)

Design & Optimize

Infra Engineer (SRE)

Provision, Monitor and Manage

**Gateway** - GraphQL | API Gateway | Gateway API | Ingress | CDN

**Applications** - Monolithic Applications | Microservices | Containers

**Services** - YugabyteDB | Messaging | Eventing | Files | Object Stores

**Runtimes** - Kubernetes | VM | PaaS

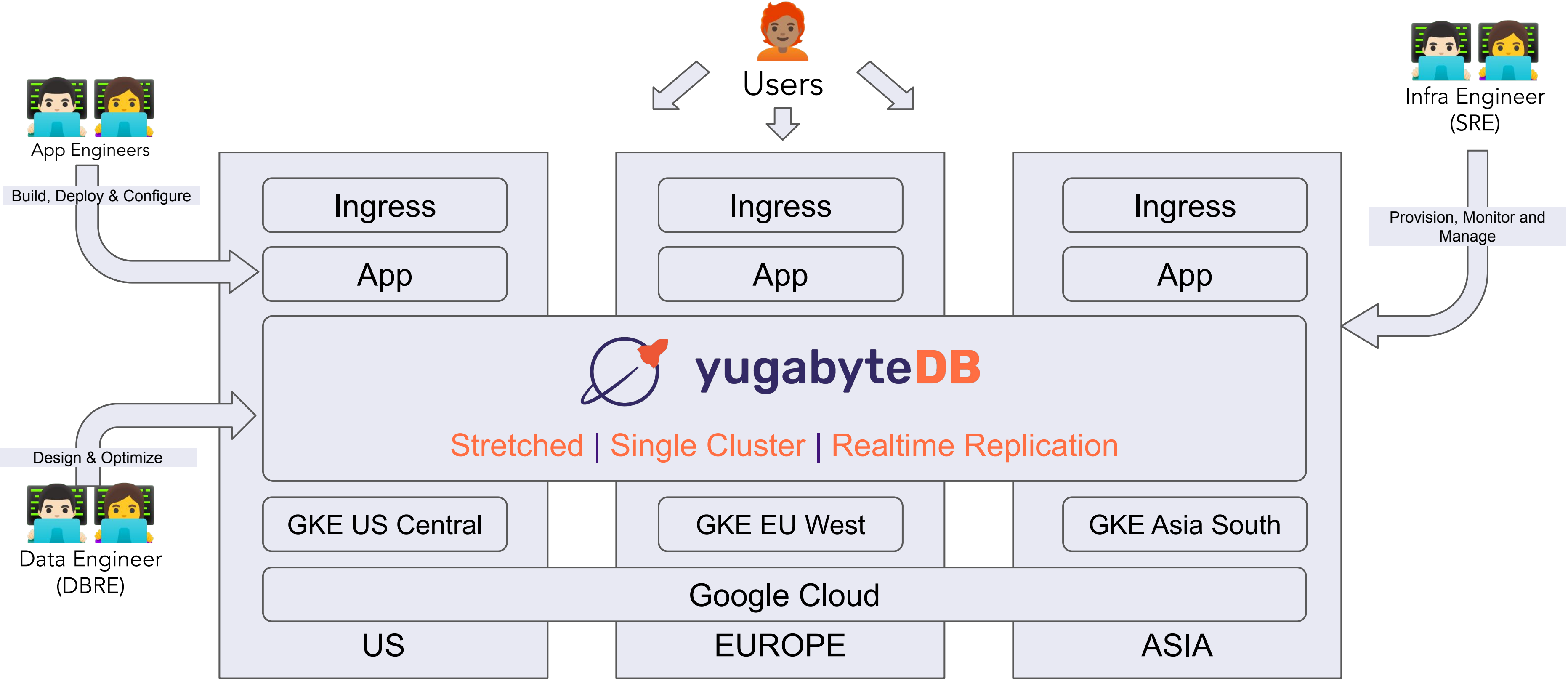
**Infrastructure** - Compute | Storage | Networking

Site 1

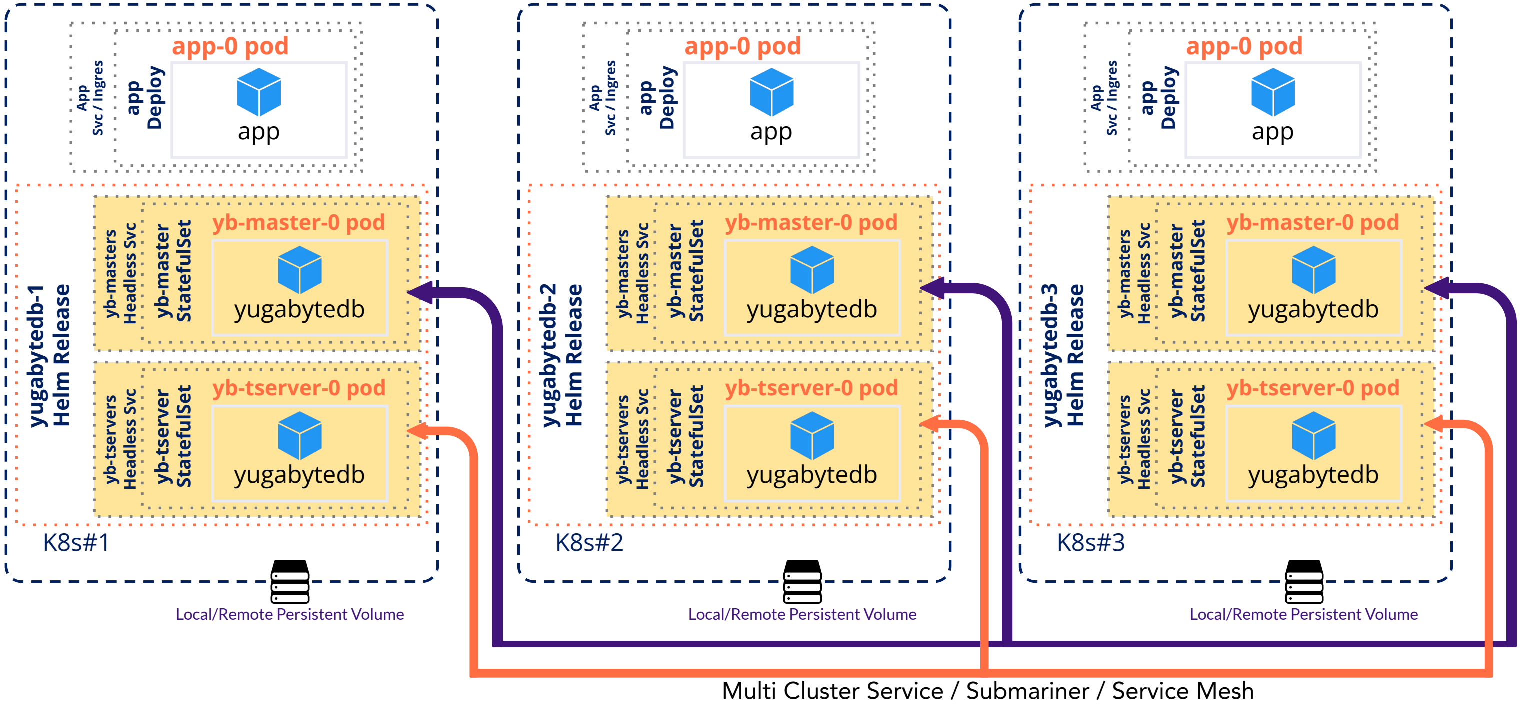
Site 2

Site 3

# Demo #2 - Target Architecture



# YugabyteDB on Multi-Kubernetes



## Demo #2: Outer Loop

# Provision Cloud Regions (3 regions)
# Deploy Database Services
# Deploy Application and Configure

**https://github.com/yogendra/yb-e2e-demo/**

# Thank You

Join us on Slack: yugabyte.com/slack

Star us on Github: github.com/yugabyte/yugabyte-db