



**By Alex Bissessur**

# Who Am I?

- Kubernetes and Cloud Native enthusiast
- Organiser of Cloud Native Mauritius
- Software Development intern at Swan
- I like Python, Rust, and Linux
- I have a small rack where I play with Kubernetes
- Aspirations?  
Play with Kubernetes for \$\$

“I do fun things with  
Kubernetes.”



# Kubernetes is easy?

```
alex@AlexArch:~  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: postgresql  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: postgresql  
  template:  
    metadata:  
      labels:  
        app: postgresql  
    spec:  
      containers:  
        - name: postgresql  
          image: postgres:latest  
          ports:  
            - containerPort: 5432  
          env:  
            - name: POSTGRES_DB  
              value: your_database_name  
            - name: POSTGRES_USER  
              value: your_username  
            - name: POSTGRES_PASSWORD  
              value: your_password  
          volumeMounts:  
            - mountPath: /var/lib/postgresql/data  
              subPath: postgresql-data  
              name: database-pvc  
      volumes:  
        - name: database-pvc  
          persistentVolumeClaim:  
            claimName: database-pvc  
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: postgresql  
spec:  
  type: NodePort  
  selector:  
    app: postgresql  
  ports:  
    - protocol: TCP  
      port: 5432  
      targetPort: 5432  
[alex@AlexArch ~]$
```

# Kubernetes is easy?

```
alex@AlexArch:~  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: postgresql  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: postgresql  
  template:  
    metadata:  
      labels:  
        app: postgresql  
    spec:  
      containers:  
        - name: postgresql  
          image: postgres:13  
          ports:  
            - containerPort: 5432  
          env:  
            - name: POSTGRES_PASSWORD  
              value: password  
            - name: POSTGRES_USER  
              value: postgres  
            - name: POSTGRES_DB  
              value: postgres  
            - name: PGDATA  
              value: /var/lib/postgresql/data  
          volumeMounts:  
            - name: postgresql-data  
              mountPath: /var/lib/postgresql/data  
      volumes:  
        - name: postgresql-data  
          persistentVolumeClaim:  
            claimName: postgresql-pvc  
---  
apiVersion: v1  
kind: PersistentVolume  
metadata:  
  name: nginx-pv  
spec:  
  capacity:  
    storage: 1Gi  
  accessModes:  
    - ReadWriteOnce  
  storageClassName: longhorn  
  persistentVolumeReclaimPolicy: Retain  
  csi:  
    driver: io.rancher.longhorn  
    fsType: ext4  
    volumeHandle: nginx-pv # This should be a unique identifier  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: nginx-pvc  
  namespace: default  
spec:  
  accessModes:  
    - ReadWriteOnce  
  resources:  
    requests:  
      storage: 1Gi  
  storageClassName: longhorn  
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: postgresql  
spec:  
  type: NodePort  
  selector:  
    app: postgresql  
  ports:  
    - protocol: TCP  
      port: 5432  
      targetPort: 5432  
[alex@AlexArch ~]$
```

# Kubernetes is easy?

```
alex@AlexArch:~$
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgresql
  template:
    metadata:
      labels:
        app: postgresql
    spec:
      containers:
        - name: postgresql
          image: postgres:latest
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_PASSWORD
              value: postgres
            - name: POSTGRES_USER
              value: postgres
            - name: POSTGRES_DB
              value: postgres
            - name: PGDATA
              value: /var/lib/postgresql/data
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgresql-storage
      volumes:
        - name: postgresql-storage
          persistentVolumeClaim:
            claimName: postgresql-pvc
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: io.rancher.longhorn
    fsType: ext4
    volumeHandle: nginx-pv
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: longhorn
---
apiVersion: v1
kind: Service
metadata:
  name: postgresql
spec:
  type: NodePort
  selector:
    app: postgresql
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
[alex@AlexArch ~]$

root@PineMaster: ~/frontendweb
# This will be the deployment setup
kind: Deployment
metadata:
  # Name your Deployment here
  name: frontendweb
  labels:
    # label your deployment
    app: frontendweb
spec:
  # The number of pods/replicas to run
  replicas: 1
  selector:
    matchLabels:
      # selector to match the pod
      app: frontendweb
  template:
    metadata:
      labels:
        # label your pod
        app: frontendweb
    spec:
      containers:
        # Add the container name for Kubernetes
        - name: frontendweb
          # Add the local image name
          image: frontendweb:latest
          # never pull the image policy
          imagePullPolicy: Never
          ports:
            - containerPort: 3000
---
# First, add the Service API
apiVersion: v1
# This will be the Service setup
kind: Service
metadata:
  # Your service name
  name: frontendweb-svc
spec:
  selector:
    # selector that matches the pod
    app: frontendweb
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  # type of service
  type: LoadBalancer
[alex@AlexArch ~]$
```



```

alex@AlexArch:~$
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgresql
  template:
    metadata:
      labels:
        app: postgresql
    spec:
      containers:
        - name: postgresql
          image: postgres:13
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_PASSWORD
              value: "postgres"
            - name: POSTGRES_USER
              value: "postgres"
            - name: POSTGRES_DB
              value: "postgres"
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgresql-data
      volumes:
        - name: postgresql-data
          persistentVolumeClaim:
            claimName: postgresql-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: postgresql-svc
spec:
  type: NodePort
  selector:
    app: postgresql
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
[alex@AlexArch ~]$

```

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: io.rancher.longhorn
    fsType: ext4
    volumeHandle: nginx-pv

```

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: longhorn

```

```

root@PineMaster:~$
# This will be the deployment setup
kind: Deployment
metadata:
  # Name your Deployment here
  name: frontendweb
  labels:
    # label your deployment
    app: frontendweb
spec:
  # The number of pods/replicas to run
  replicas: 1
  selector:
    matchLabels:
      # selector to match the pod
      app: frontendweb
  template:
    metadata:
      labels:
        # label your pod
        app: frontendweb
    spec:
      containers:
        # Add the container name for Kube
        - name: frontendweb
          # Add the local image name
          image: frontendweb:latest
          # never pull the image policy
          imagePullPolicy: Never
          ports:
            - containerPort: 3000

```

```

---
# First, add the Service API
apiVersion: v1
kind: Service
metadata:
  # This will be the Service setup
  name: frontendweb-svc
spec:
  # Your service name
  name: frontendweb-svc
  selector:
    # selector that matches the pod
    app: frontendweb
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  # type of service
  type: LoadBalancer

```

```

root@PineMaster: ~/ghost/new
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ghost-deployment
spec:
  replicas: 1 # You can adjust the number of replicas as needed
  selector:
    matchLabels:
      app: ghost
  template:
    metadata:
      labels:
        app: ghost
    spec:
      containers:
        - name: ghost
          image: ghost:5.71.0
          ports:
            - containerPort: 2368
          volumeMounts:
            - name: ghost-data
              mountPath: /bitnami/ghost
          env:
            - name: ALLOW_EMPTY_PASSWORD
              value: "yes"
            - name: GHOST_DATABASE_HOST
              value: mysql
            - name: GHOST_DATABASE_PORT_NUMBER
              value: "3306"
            - name: GHOST_DATABASE_USER
              value: bn_ghost
            - name: GHOST_DATABASE_NAME
              value: bitnami_ghost
          volumes:
            - name: ghost-data
              persistentVolumeClaim:
                claimName: ghost-data # Replace with your PVC name

```

```

---
apiVersion: v1
kind: Service
metadata:
  name: ghost-service
spec:
  selector:
    app: ghost
  ports:
    - protocol: TCP
      port: 80
      targetPort: 2368

```

easy?

```

alex@AlexArch:~$
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgresql
  template:
    metadata:
      labels:
        app: postgresql
    spec:
      containers:
        - name: postgresql
          image: postgres:13
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_PASSWORD
              value: "postgres"
            - name: POSTGRES_USER
              value: "postgres"
            - name: POSTGRES_DB
              value: "postgres"
      volumes:
        - name: postgresql-data
          persistentVolumeClaim:
            claimName: postgresql-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: postgresql
spec:
  type: NodePort
  selector:
    app: postgresql
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
[alex@AlexArch ~]$

```

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: io.rancher.longhorn
    fsType: ext4
    volumeHandle: nginx-pv

```

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: longhorn

```

```

[alex@AlexArch ~]$

```

```

root@PineMaster:~$
# This will be the deployment setup
kind: Deployment
metadata:
  # Name your Deployment here
  name: frontendweb
  labels:
    # label your deployment
    app: frontendweb
spec:
  # The number of pods/replicas to run
  replicas: 1
  selector:
    matchLabels:
      # selector to match the pod
      app: frontendweb
  template:
    metadata:
      labels:
        # label your pod
        app: frontendweb
    spec:
      containers:
        # Add the container name for Kube
        - name: frontendweb
          # Add the local image name
          image: frontendweb:latest
          # never pull the image policy
          imagePullPolicy: Never
          ports:
            - containerPort: 3000

```

```

---
# First, add the Service API
apiVersion: v1
kind: Service
metadata:
  # This will be the Service setup
  name: frontendweb-svc
spec:
  # Your service name
  name: frontendweb-svc
  selector:
    # selector that matches the pod
    app: frontendweb
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  # type of service
  type: LoadBalancer

```

```

root@PineMaster: ~/ghost/new
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ghost-deployment
spec:
  replicas: 1 # You can
  selector:
    matchLabels:
      app: ghost
  template:
    metadata:
      labels:
        app: ghost
    spec:
      containers:
        - name: ghost
          image: ghost:5.
          ports:
            - containerPort: 2368
      volumeMounts:
        - name: ghost-data
          mountPath: /bitnami/ghost
      env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        - name: GHOST_DATABASE_HOST
          value: mysql
        - name: GHOST_DATABASE_PORT_NUMBER
          value: "3306"
        - name: GHOST_DATABASE_USER
          value: bn_ghost
        - name: GHOST_DATABASE_NAME
          value: bitnami_ghost
      volumes:
        - name: ghost-data
          persistentVolumeClaim:
            claimName: ghost-data # Replace with your PVC name
---
apiVersion: v1
kind: Service
metadata:
  name: ghost-service
spec:
  selector:
    app: ghost
  ports:
    - protocol: TCP
      port: 80
      targetPort: 2368

```

```

root@PineMaster:~/metallb# cat ipaddress_pools.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: production
  namespace: metallb-system
spec:
  addresses:
    - 192.168.1.210-192.168.1.250
root@PineMaster:~/metallb#

```

```

alex@AlexArch:~
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgresql
  template:
    metadata:
      labels:
        app: postgresql
    spec:
      containers:
        - name: postgresql
          image: postgres:13
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_PASSWORD
              value: postgres
            - name: POSTGRES_USER
              value: postgres
            - name: POSTGRES_DB
              value: postgres
            - name: PGDATA
              value: /var/lib/postgresql/data/pgdata
          volumeMounts:
            - name: postgresql-storage
              mountPath: /var/lib/postgresql/data
      volumes:
        - name: postgresql-storage
          persistentVolumeClaim:
            claimName: postgresql-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgresql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: longhorn
---
apiVersion: v1
kind: Service
metadata:
  name: postgresql
spec:
  type: NodePort
  selector:
    app: postgresql
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
[alex@AlexArch ~]$

```

```

root@PineMaster: ~
# This will be the deployment setup
kind: Deployment
metadata:
  # Name your Deployment here
  name: frontendweb
  labels:
    # label your deployment
    app: frontendweb
spec:
  # The number of pods/replicas to run
  replicas: 1
  selector:
    matchLabels:
      # selector to match the pod
      app: frontendweb
  template:
    metadata:
      labels:
        # label your pod
        app: frontendweb
    spec:
      containers:
        # Add the container name for Kube
        - name: frontendweb
          # Add the local image name
          image: frontendweb:latest
          # never pull the image policy
          imagePullPolicy: Never
          ports:
            - containerPort: 3000
---
# First, add the Service API
apiVersion: v1
kind: Service
metadata:
  # This will be the Service setup
  name: frontendweb-svc
spec:
  # Your service name
  name: frontendweb-svc
  selector:
    # selector that matches the pod
    app: frontendweb
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  # type of service
  type: LoadBalancer

```

```

root@PineMaster: ~/ghost/new
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ghost-deployment
spec:
  replicas: 1 # You can
  selector:
    matchLabels:
      app: ghost
  template:
    metadata:
      labels:
        app: ghost
    spec:
      containers:
        - name: ghost
          image: ghost:5.
          ports:
            - containerPort: 2368
          volumeMounts:

```

```

root@PineMaster:~/metallb# cat ipaddress_pools.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: production
  namespace: metallb-system
spec:
  addresses:
    - 192.168.1.210-192.168.1.250
root@PineMaster:~/metallb#

```

```

root@PineMaster: ~/longhorn-storage
apiVersion: v1
kind: PersistentVolume
metadata:
  name: storage-pv
spec:
  capacity:
    storage: 60Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /storage
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - potato-7

```

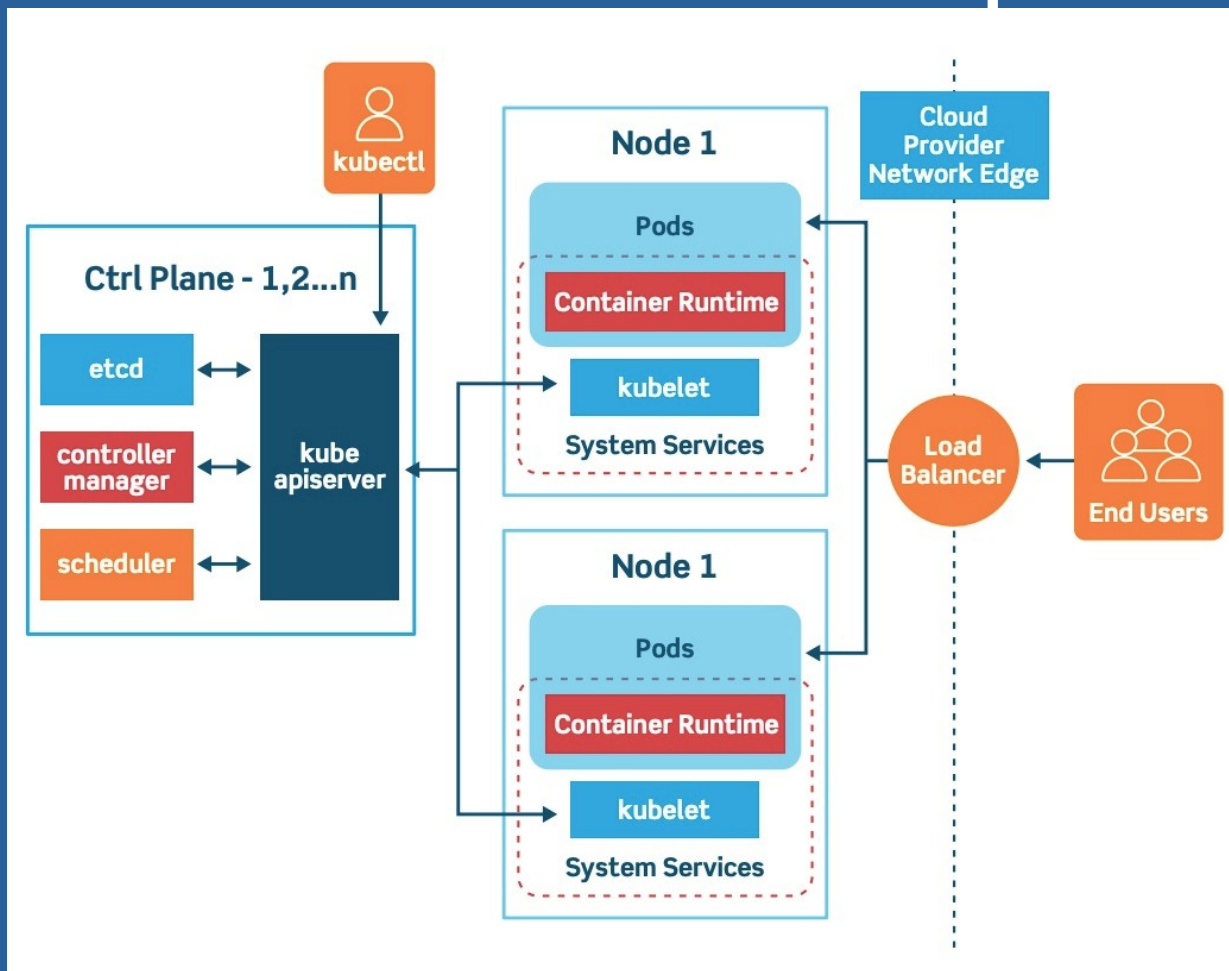
```

root@PineMaster:~/longhorn-storage#

```



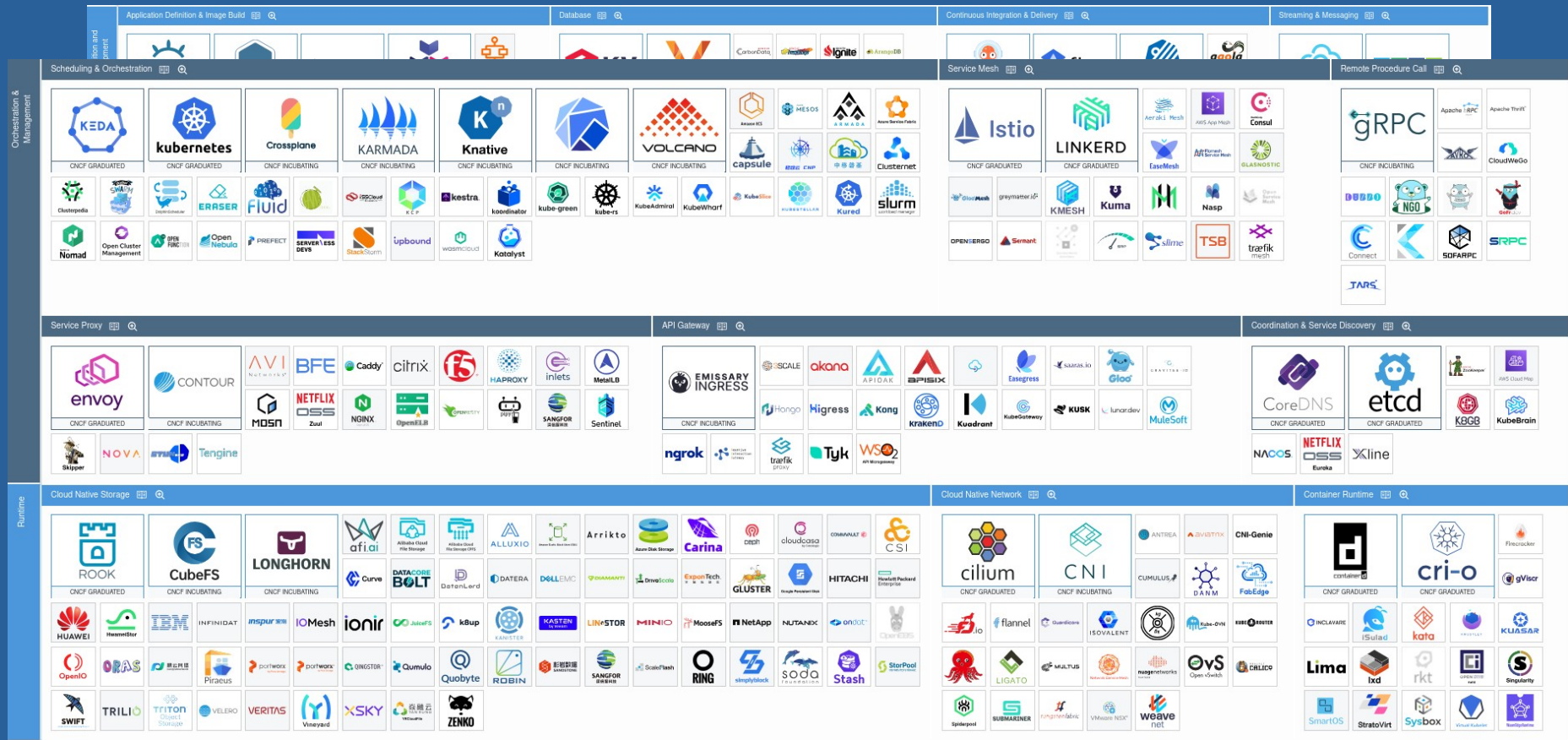
# Kubernetes is simple?



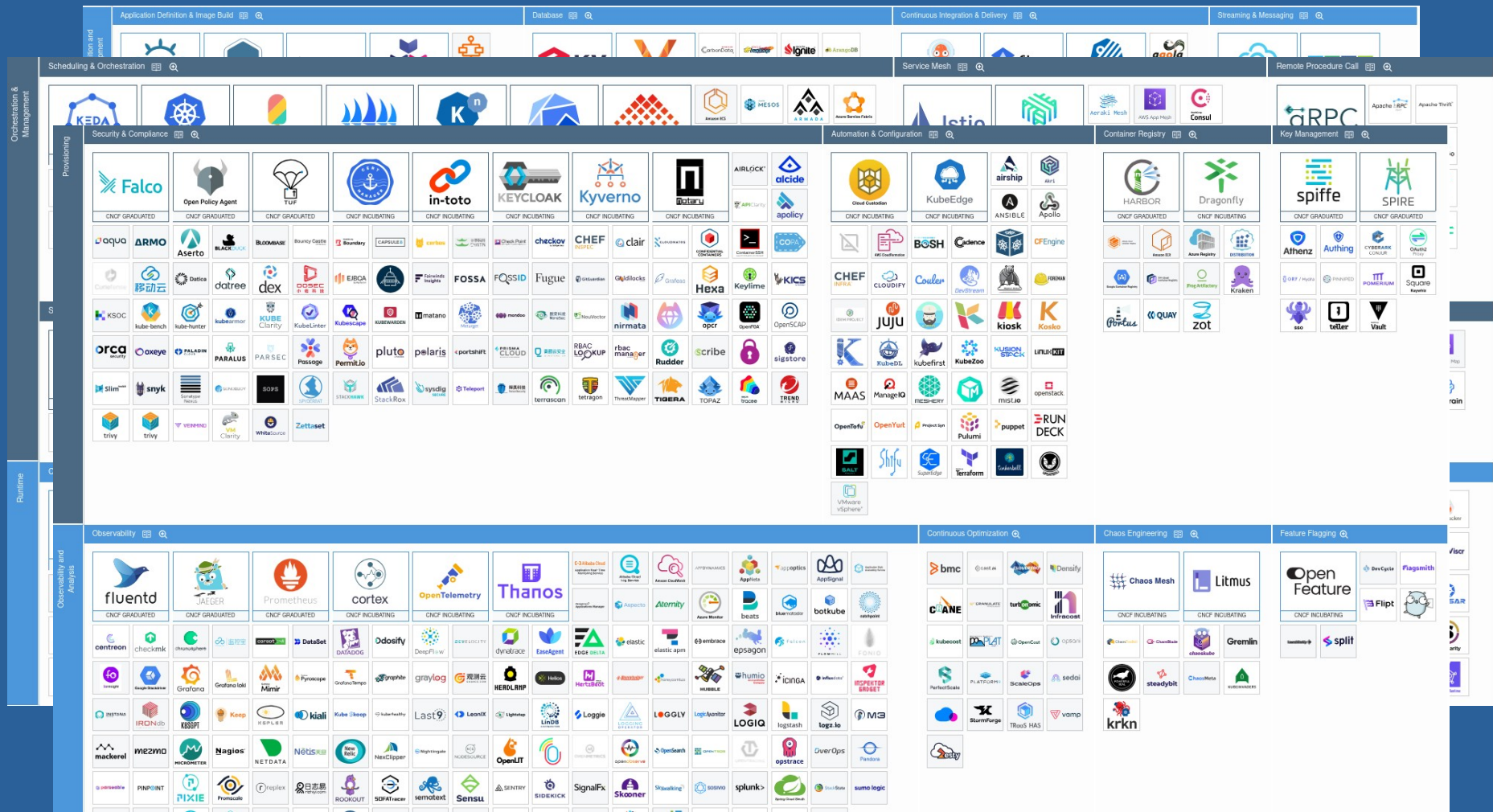
The diagram displays a comprehensive collection of open-source project logos, organized into five main functional categories:

- Application Definition & Image Build**: Includes Helm, Artifact Hub, Backstage, Buildpacks.io, CARVEL, Dapr, KubeVela, KubeVirt, OPERATOR FRAMEWORK, KubePlus, and many others.
- Database**: Features KV, Vitess, CarbonDB, BigchainDB, Cockroach Labs, Couchbase, CRUX, Datastax, Dgraph, Doris, Druid, EDB, Epilila, Flink, Gearspring, HBase, InfluxDB, Iguazio, Jooq, Kudu, MariaDB, MySQL, Neo4j, Oracle, PostgreSQL, Redis, SAP, Seata, SingleStore, Snowflake, Stolon, TIDB, Timescale, UDB, Vertica, VoltDB, Weaviate, YDB, YugabyteDB, and PolarDB.
- Continuous Integration & Delivery**: Contains Argo, Flux, Keptn, Agola, OpenKruise, Skycap, CloudBees, Codefresh, Concourse, CircleCI, GitLab, GitHub Actions, Jenkins, K6, Kephloy, Liquibase, Merigify, Northbank, OpenGitOps, Psmix, Quxone, Razer, Semaphore, Spacelift, Sprinklr, TeamCity, Tekton, Travis CI, Unleash, Werf, and others.
- Streaming & Messaging**: Includes Cloudevents, NATS, STRIMZI, NiFi, Spark, Storm, Strimzi, Telemetry, Beam, CD Events, EMQ, Flink, Fluvio, Kafka, KuberMQ, Akka, Memphis, Pulsar, RabbitMQ, Redpanda, Siddhi, Talend, Tremor, and AutoMQ.
- Other Categories**: A section labeled "Others" containing logos like Ansible, Apache Airflow, Apache Beam, Apache Camel, Apache Flink, Apache Kafka, Apache Kudu, Apache Spark, Apache Storm, Apache Tez, Apache Zeppelin, AWS Lambda, Azure Functions, Google Cloud Functions, IBM Watson, Microsoft Azure, Oracle Cloud, Salesforce, SAP, ServiceNow, Slack, Stripe, Twilio, Uber, and many more.

# Kubernetes is straightforward?



# Kubernetes is straightforward?





# What Is Kubernetes?

- “K8S is a system for automating deployment, scaling and management of containerised applications”
  - Simply an abstraction/framework to run a cluster of computers
- Industry standard with a whole ecosystem behind
- Platform for 200~ Cloud Native Applications
- Former Google product, now Open Source held by CNCF (Linux Foundation project)



# Why is Kubernetes a Thing?

- Need for cluster computing solutions which don't suck ( Docker Swarm 🤢 )
- High Availability, Redundancy, Easy to Manage
- CSP agnostic (AWS / Azure / GCP / Bare Metal)
- Offers a platform suited to both admins and devs

# Kubernetes Components Simplified

- Nodes
- Pods
- ReplicaSets
- Deployments
- Namespaces
- Services
  - NodePort
  - LoadBalancer
  - ClusterIP
- Persistent Volumes
- Persistent Volume Claims

and many more...

# Pod

- The smallest unit in Kubernetes
- May contain 1 or more containers
- Ephemeral – can be created, destroyed, and recreated dynamically
- No self-healing or scaling capabilities
- Docker run – but complicated

# Deployment

- Encapsulates Pods through ReplicaSets
- Ensures desired state is maintained (eg replicas)
- Self Healing
- Scalability
- Revision history for rollbacks
- Powerful update strategies  
(RollingUpdate, Recreate, etc...)

# Service

- How do you access pods regardless of where they are in the cluster?
- Services provide constant routes to variable pods through internal networking
- Different types for different traffic types (internal, external)
- Supports load balancing
- Detailed control over traffic routing policies



# Service

- ClusterIP – communication inside cluster  
uses: Constant routing within cluster
- NodePort – exposing app through port on cluster nodes  
uses: Exposing application through dedicated port
- LoadBalancer – offered by CSPs  
uses: Dedicated IP for application running on K8S

# Persistent Volume (Claim)

- Abstraction for persistent volume in Kubernetes
- Deployments are stateless
  - Restart container = drop tables
- PV can be NFS, iSCSI, or even S3 compatible
- PVC is an interface for PVs
- PVC types
  - Read (Only/Write) Many
  - Read Write Once (Node/Pod)

# Why Use Kubernetes

- Single PC has limited upgradeability
- Expand horizontally not vertically  
Quantity over quality
- Redundancy
- Exploiting microservices and containers
- Better upgradeability, availability, portability

# Building on Kubernetes

- The power of Kubernetes is the CN Landscape
- ~ 200 Cloud Native projects and counting

- CI/CD
- App Definition & Image Build
- Databases
- Streaming & Messaging
- Scheduling & Orchestration
- API Gateway
- Discovery
- Cloud Native Storage
- Cloud Native Network
- Service Mesg
- Remote Procedure Call
- Service Proxy
- Chaos Engineering
- Continuous Optimisation
- Container Runtime
- Security & Compliance
- Container Registry
- Automation & Configuration
- Key Management
- Observability
- Feature Flagging
- Coordination & Service

# Physical Scaling

- One server cannot be upgraded indefinitely
- Physical limitations in compute power
- Instead of 1 server, why not have 10?  
Kubernetes unifies multiple servers
- Kubernetes spreads pods across all nodes to best use resources



# Virtual Scaling

- Scaling microservices  
Possible due to lack of state
- Allows optimal use of resources
- Scale up/down to accomodate demand
- Horizontal Pod Autoscaler for hands-off scaling
- Automatic load balancing of applications

# Redundancy

- 1 server = 1 point of failure
- Self-healing of deployments in case of failure
  - Pods independent from nodes
  - Services “follow” pods
- Longhorn/Rook+Ceph allows distributed block storage for data redundancy
- Multi-cloud cluster for added redundancy
- Benefits physical upgrades or OS updates

# Easy Management

- Containerised means consistent between local, test & prod
- GitOps and CI/CD pipelines for automatic deployment
- No need to beg someone to deploy your app  
apologies to my colleagues
- Simpler scaling and resource management
  - Can ensure my app doesn't eat all the RAM

# Easy Management

Application on localhost



Status 200  
Success Response

Same code when deployed



Cors Error  
Cookies Sharing issues  
Bad Request

# Fancy Deployment Strategies

- Deployments can have different rollout strategies
- Algorithms like *Canary* deployment, *Rolling* update
- Blue/Green deployment concept
- Detailed control over rollout to minimise customer impact



# Rolling Update

- Starts a new pod
- Once new pod is running, takes down one old
- Number upgraded at a time can be set
- Works through all pods until they are all up to date
- Traffic loadbalanced to both old & new pods

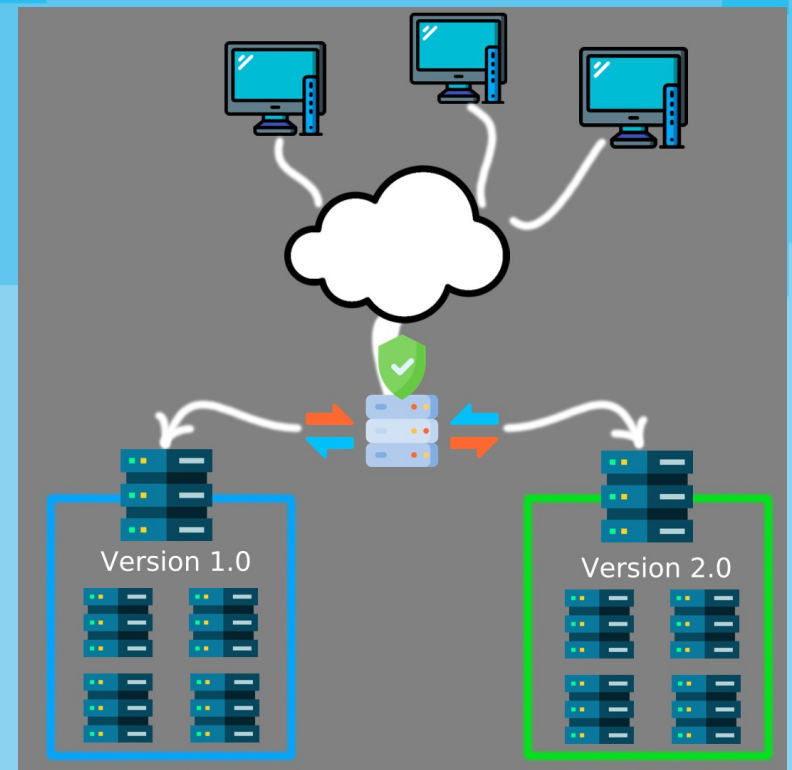
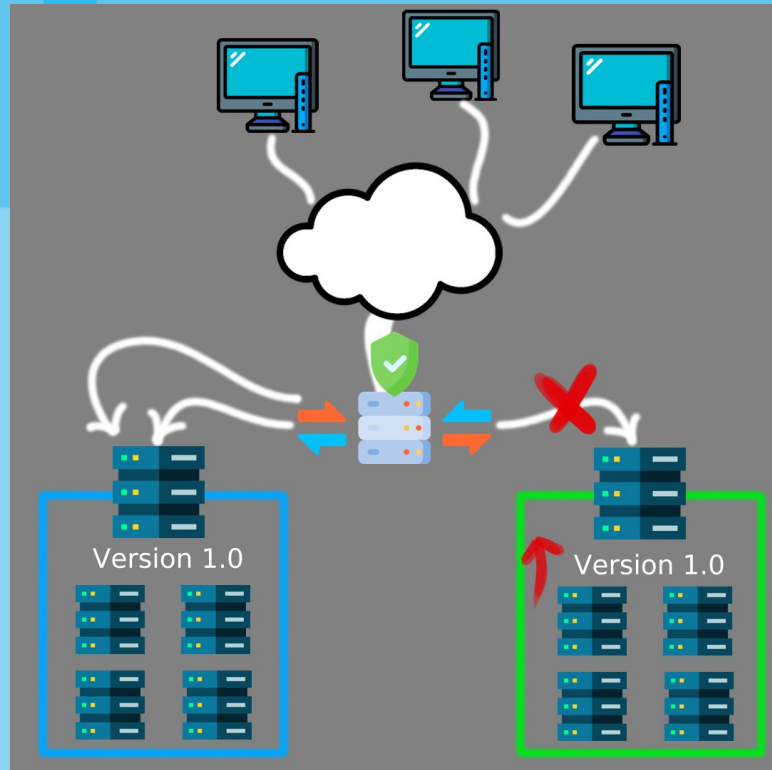
# Canary Deployment

- Initially deploy a small number of new pods
- Gradually increase traffic to latest set
- In case of errors, deployment automatically rolled back
- Requires additional CN tools, such as ArgoCD, Istio or Linkerd

# Blue-Green Deployment

- 2 environments created – Blue and Green
- All traffic routed to Blue
- Green group is upgraded and tested
- Traffic rerouted to Green
- Easy reroute back to Blue if Green has issues
- Upgrade Blue in turn
- 0 downtime, consistent application for users

# Blue-Green Deployment



# Shadow Deployment

- Create a new mirrored deployment
- Traffic is mirrored to new version
- Both versions process traffic
- Only Old returns responses
- Allows live testing with real world data  
(now you can test on “prod”)

# What's not so cool about Kubernetes

- Cost
- Expertise
- Keep it up to date
  - K8S has support for last 3 versions
  - Tool/plugin dependencies
- Steep learning curve
- Legacy monolith projects
  - Not cloud-optimised, lose out on cost benefits

# Thank You!



[alexbissessur.dev](https://alexbissessur.dev)

[moris.social/@AlexB](https://moris.social/@AlexB)