Container Orchestration (with Kubernetes) ~~Marketing~~

Peter Chen

About Me

- 2014-2016: grad student at UBC (Ivan was my supervisor)
- 2016-2019: Arista Networks
- 2019-Present: Google

Container Orchestration

- What are containers?

- What is container orchestration? What problem does it solve?

- How it relates to concepts you've learned in distributed systems?

Container Orchestration - Why we care?

- 2,429 companies use Kubernetes, a container orchestration system we will talk about, including (Google, Facebook, Shopify, …) to run their internal systems and to power their Cloud offerings
    - https://stackshare.io/kubernetes

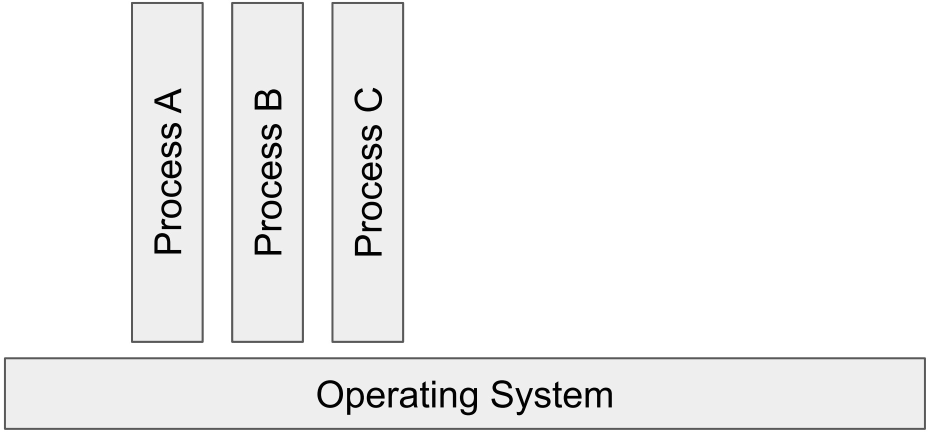
- Large open source community, as well as backed by big companies such as Docker and Google

- Gartner report adoption of containers grew 40% in 2020, by 2023, 70% of **all** organizations will be running containers in some form (in 2019 this was less than 20%)
    - Become the de-facto way to deploy, run and build services

Reminder: Processes

Operating System

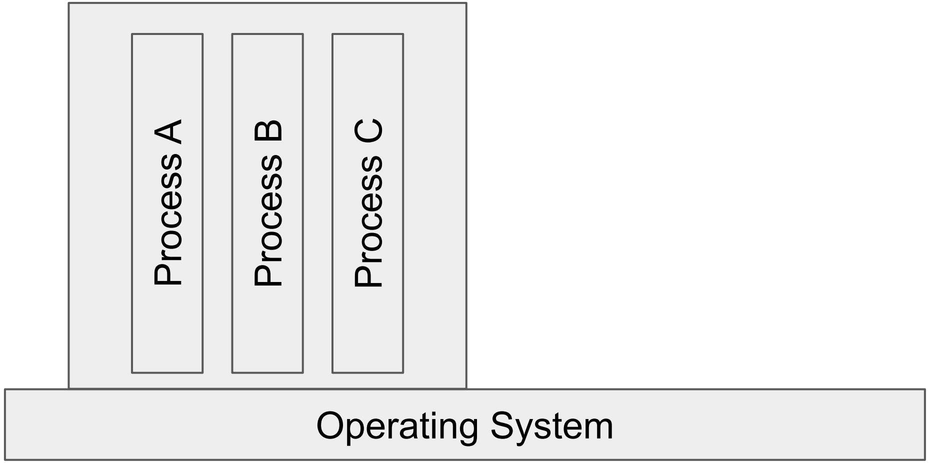
Reminder: Processes



Applications run as the **process** abstraction

Each process has its own memory space and therefore its own execution context

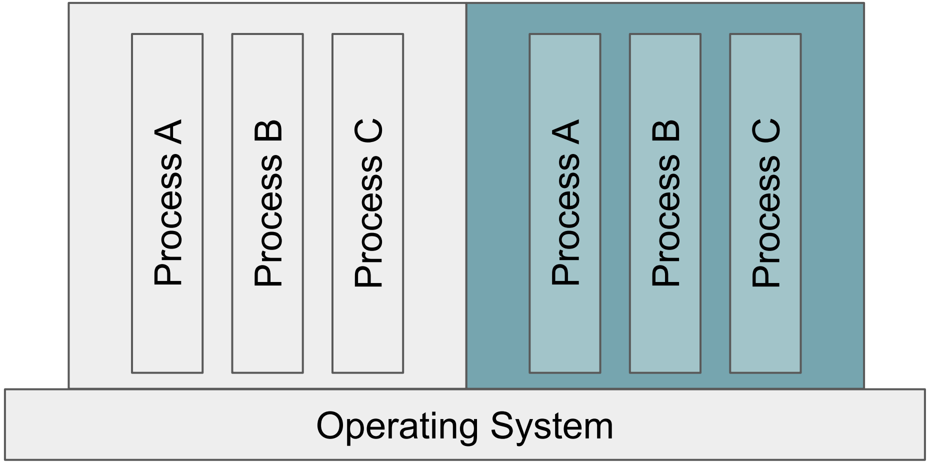
More Info: Processes



But there are other things these process share in the operating system:

- PID (process ids)
- MNT (file system)
- IPC (sockets)
- UTS (time?)
- NET (network e.g., IP tables)
- etc,...

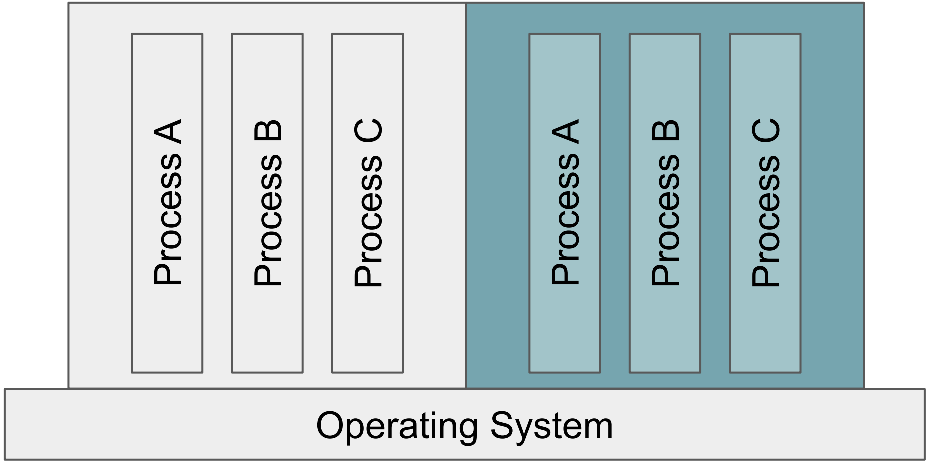
Containers



Nice things about containers, they are:

- **Lightweight**: fast, very little overhead
- **Isolation:** executable package of software with its own code, runtime, tools, libraries and settings
- **Portable:** compiled into an "image" which can be deployed on other machines as a "container" instance

Containers

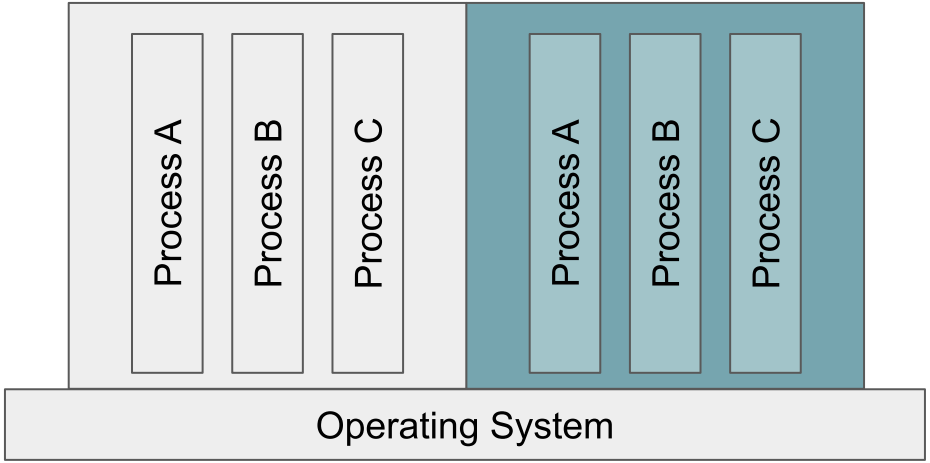


Nice things about containers, they are:

- **Lightweight**: fast, very little overhead
- **Isolation:** executable package of software with its own code, runtime, tools, libraries and settings
- **Portable:** compiled into an "image" which can be deployed on other machines as a "container" instance

This is a bit different to what you are use to in VMs where the objective is virtualization of hardware resources instead of just isolation of all the aspects of execution.

Containers

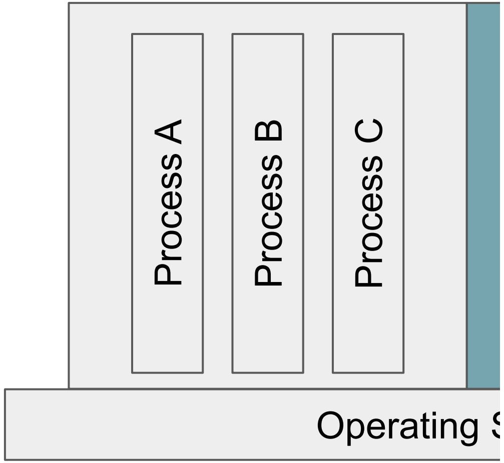


Nice things about containers, they are:

- **Lightweight**: fast, very little overhead
- **Isolation:** executable package of software with its own code, runtime, tools, libraries and settings
- **Portable:** compiled into an "image" which can be deployed on other machines as a "container" instance

Cost of containers: 5 MB for smallest image, arbitrary amount of CPU
Cost of VMs: (~2GB) for smallest OS, compute cost in increments of 1 CPU

Containers



t containers, they are:

ht: fast, very little overhead
executable package of
ith its own code, runtime,
ies and settings
compiled into an image

This is a bit differ
objective is virtualization of hardware resources instead of just
isolation of all the aspects of execution.

Container Orchestration

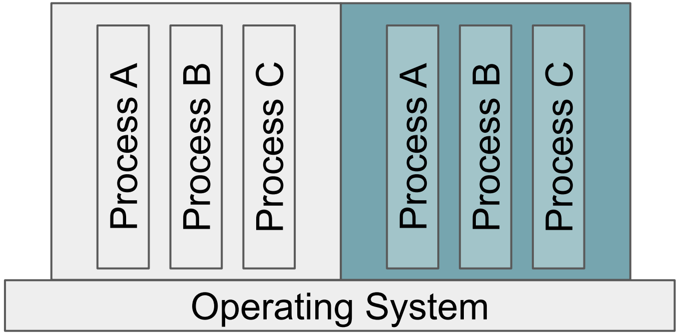
Recap:

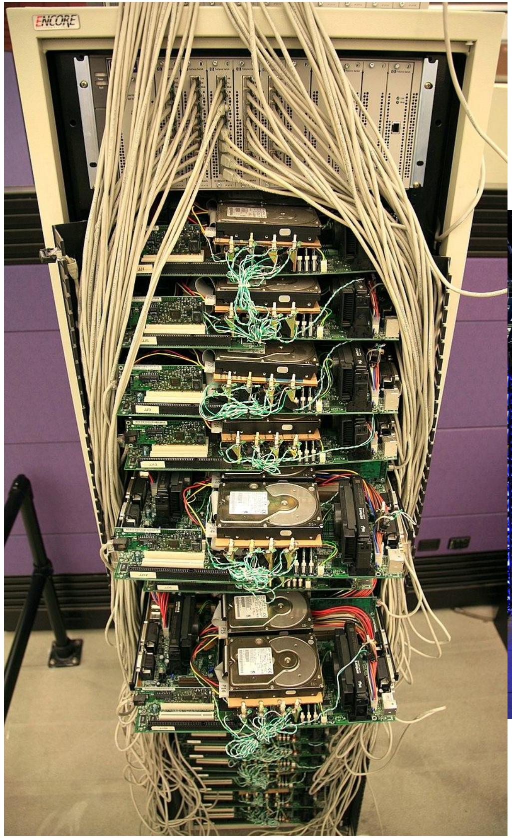
- What is the relationship between an “image” and a “container”?

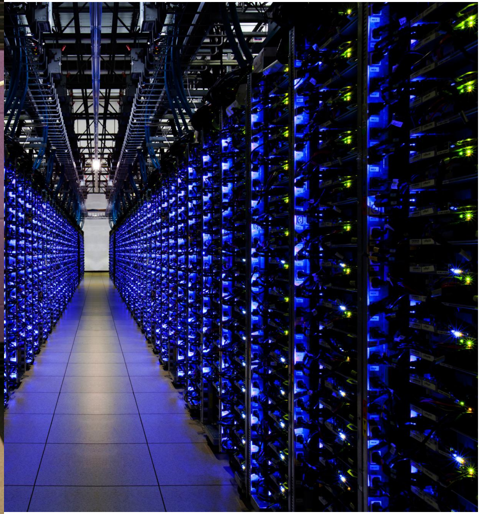
- What does running in a container isolate vs. say a VM?

- What are some of the benefits?

Container Orchestration

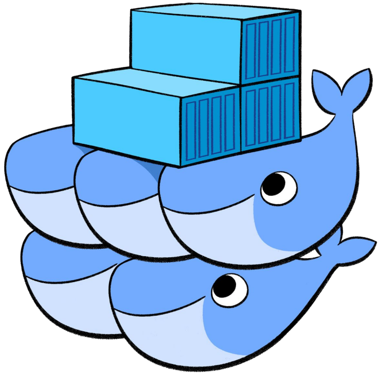




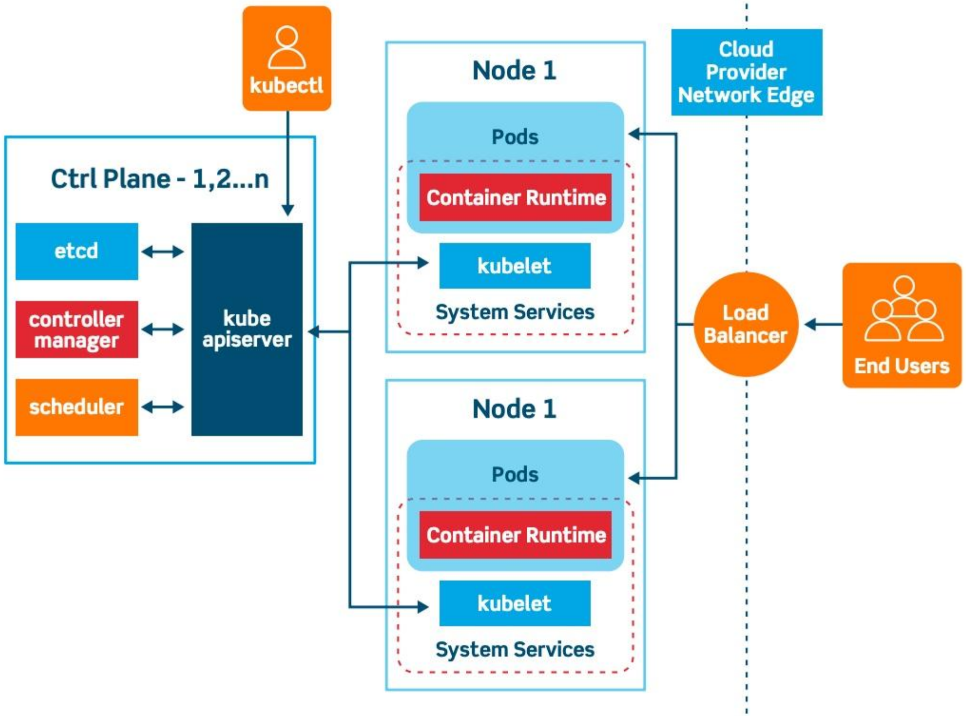


Kubernetes

kubernetes

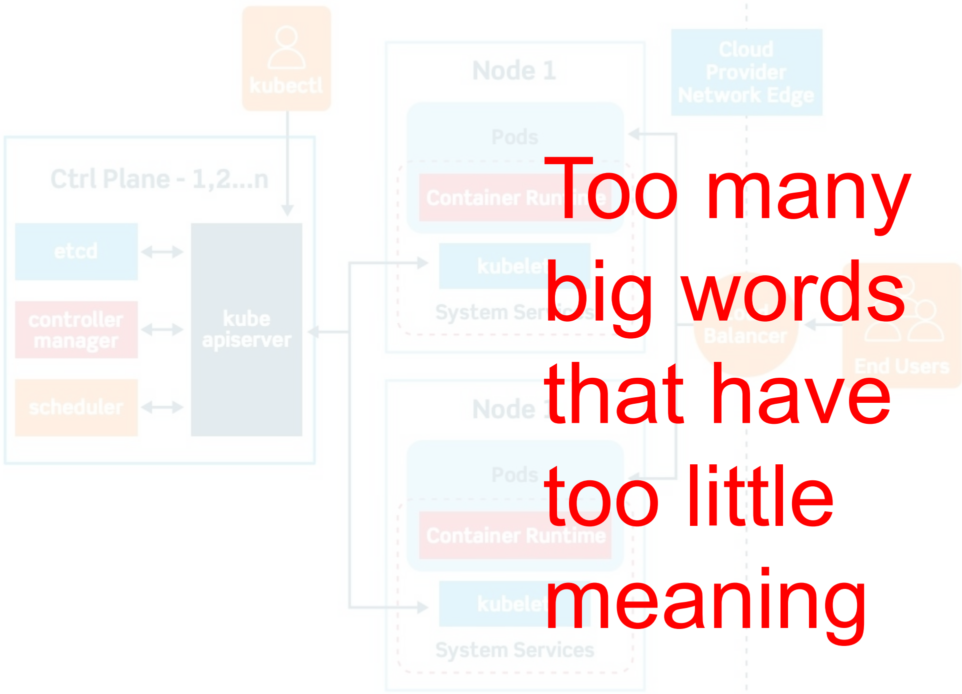


Kubernetes - Architecture



- Abstractions: Pods, Services, Ingress, Deployments, Volumes…
- Add-ons: Istio, Knative...
- Network: Weave, Flannel…
- Control Plane: scheduler, api-server, controller-manager, etcd...

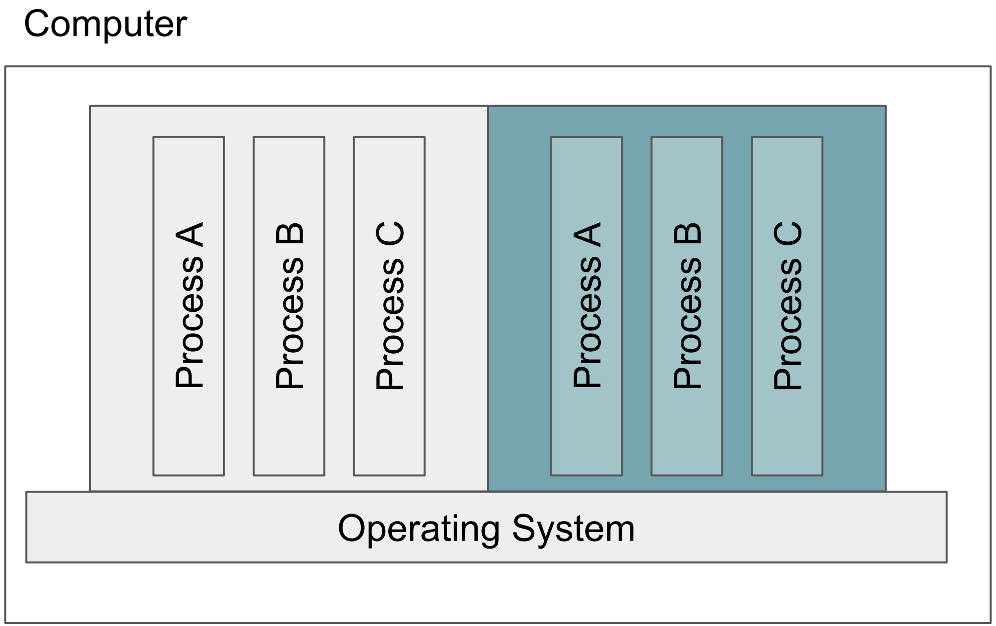
Kubernetes - Architecture



Too many big words that have too little meaning

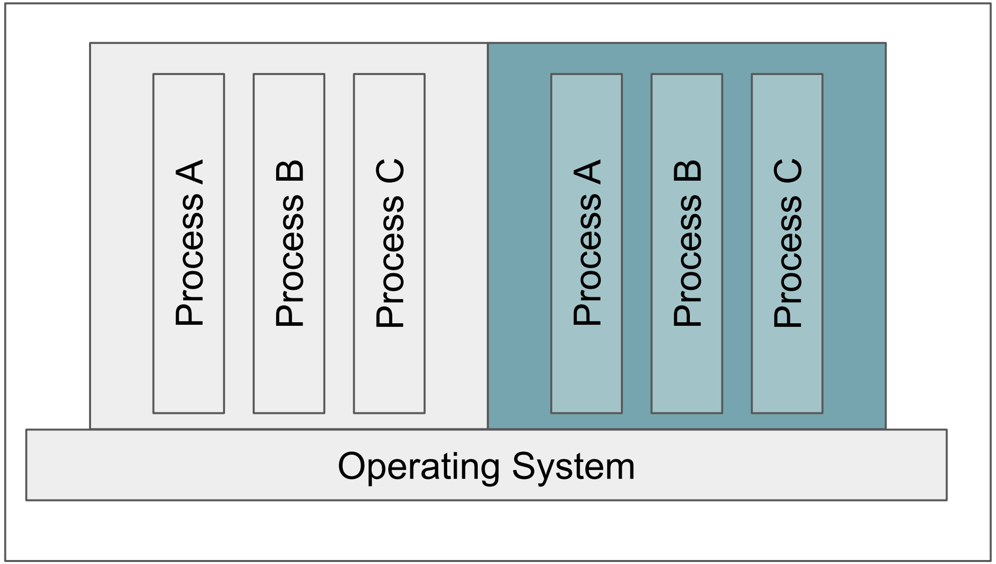
- Abstractions: Pods, Services, Ingress, Deployments, Volumes…
- Add-ons: Istio, Knative...
- Network: Weave, Flannel…
- Control Plane: scheduler, api-server, controller-manager, etcd...

Kubernetes - Execution



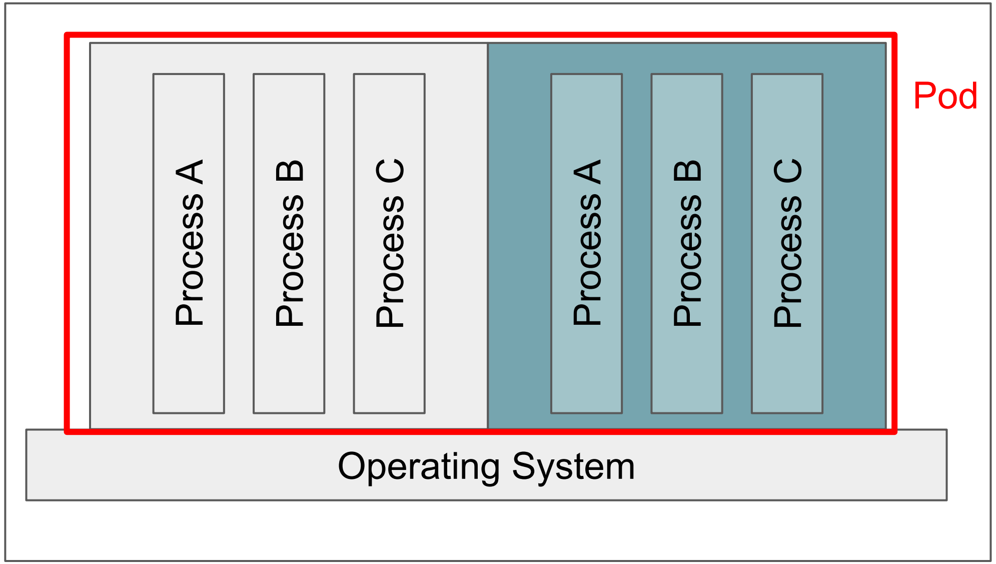
Kubernetes - Execution

~~Computer~~ **Node**

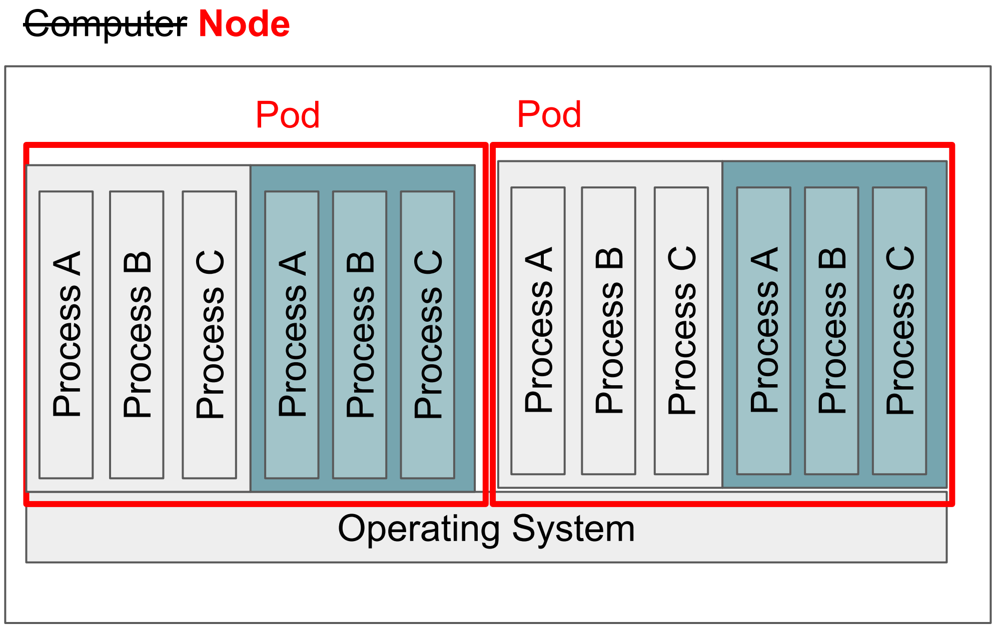


Kubernetes - Execution

~~Computer~~ **Node**

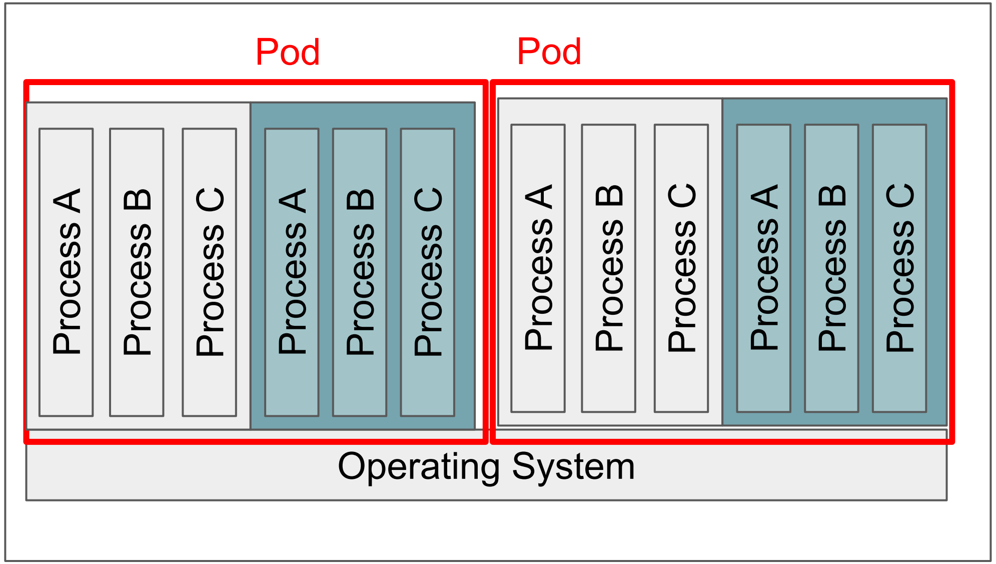


Kubernetes - Execution



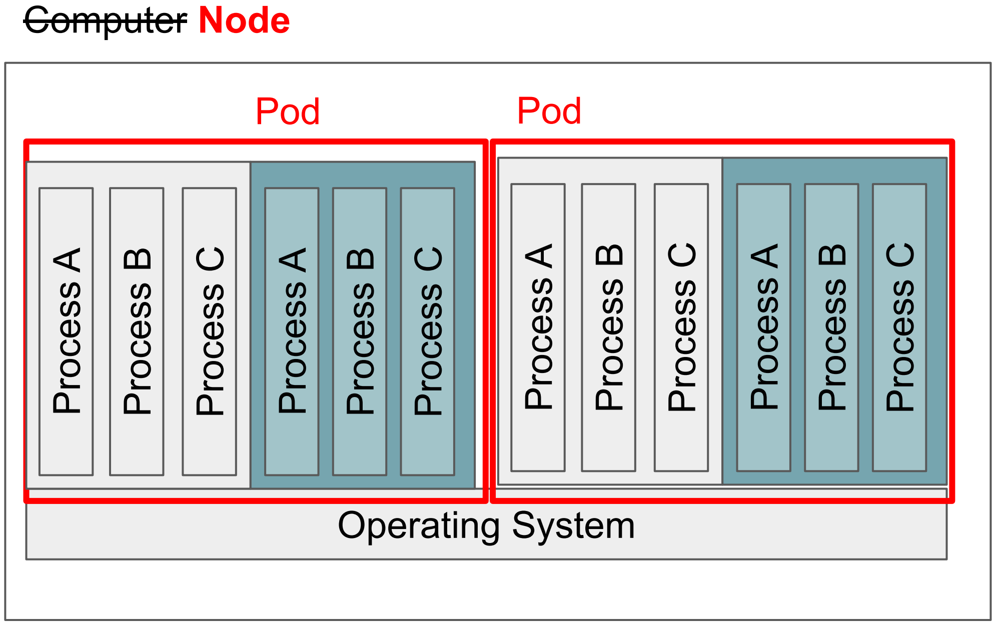
Kubernetes - Execution

~~Computer~~ **Node**



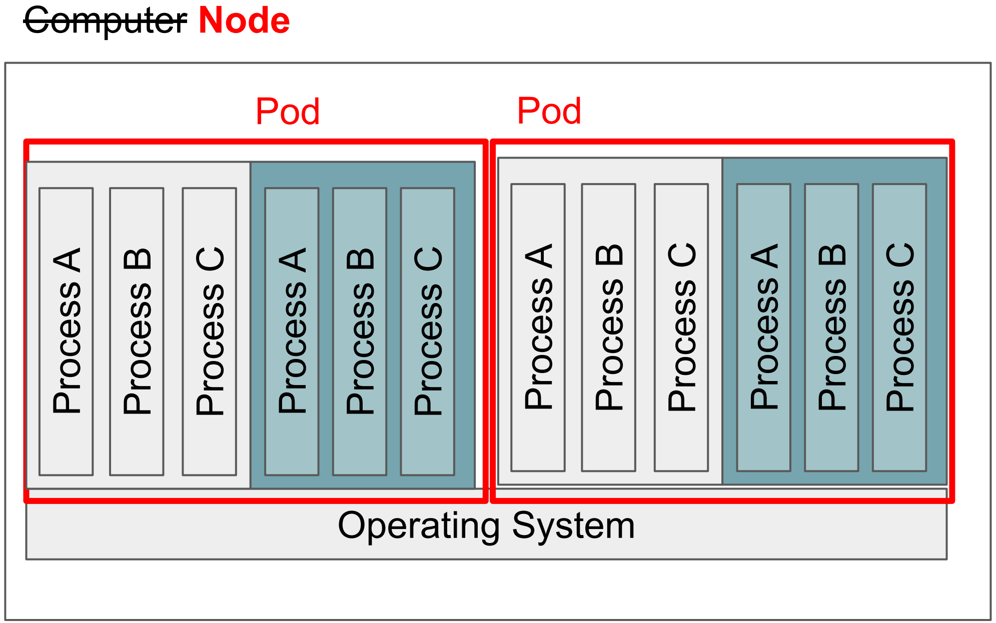
- **Fate sharing:** e.g., processes in a **pod** live/die together (a webserver and its local SQL instance

Kubernetes - Execution



- **Fate sharing:** e.g., processes in a **pod** live/die together (a webserver and its local SQL instance
- **Fault tolerance and scalability:** multiple **pods** can become ***replicas*** and execute on different **nodes** dynamically

Kubernetes - Execution

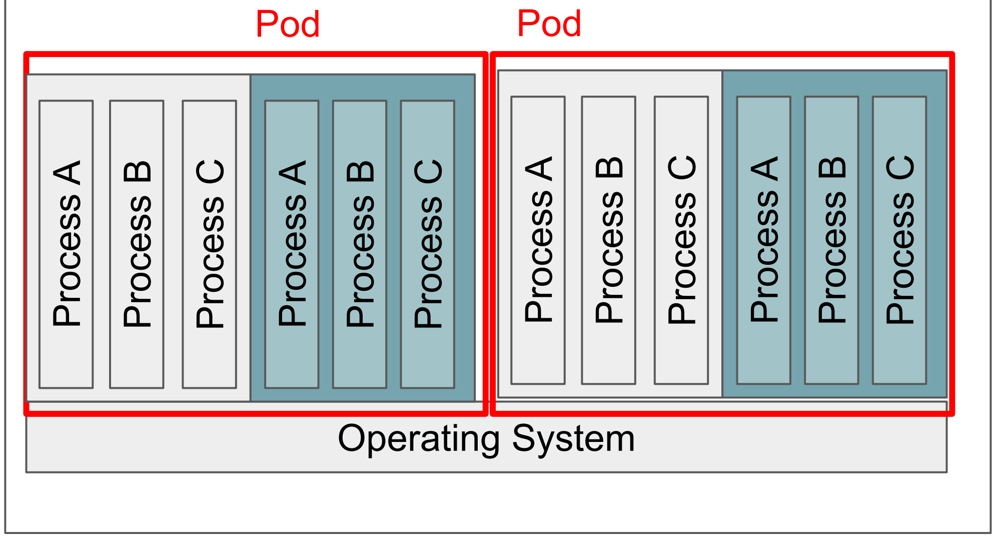


- **Fate sharing:** e.g., processes in a **pod** live/die together (a webserver and its local SQL instance
- **Fault tolerance and scalability:** multiple **pods** can become ***replicas*** and execute on different **nodes** dynamically
- **Multi-tenancy:** multiple **pods** can run on a single **node**, provide scalability and efficient use of cluster-wide resources

Kubernetes - Execution

Question: For Cloud Providers, why is resource efficiency related to availability?

~~Computer~~ **Node**

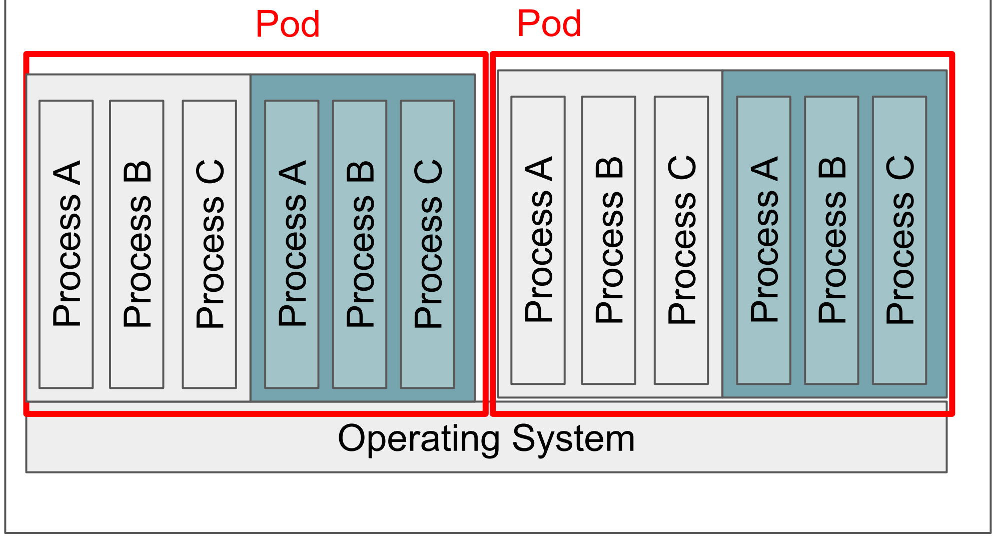


- **Fate sharing:** e.g., processes in a **pod** live/die together (a webserver and its local SQL instance
- **Fault tolerance and scalability:** multiple **pods** can become ***replicas*** and execute on different **nodes** dynamically
- **Multi-tenancy:** multiple **pods** can run on a single **node**, provide scalability and efficient use of cluster-wide resources

Kubernetes - Execution

Do you manually create/delete pods? That would be a huge hassle.

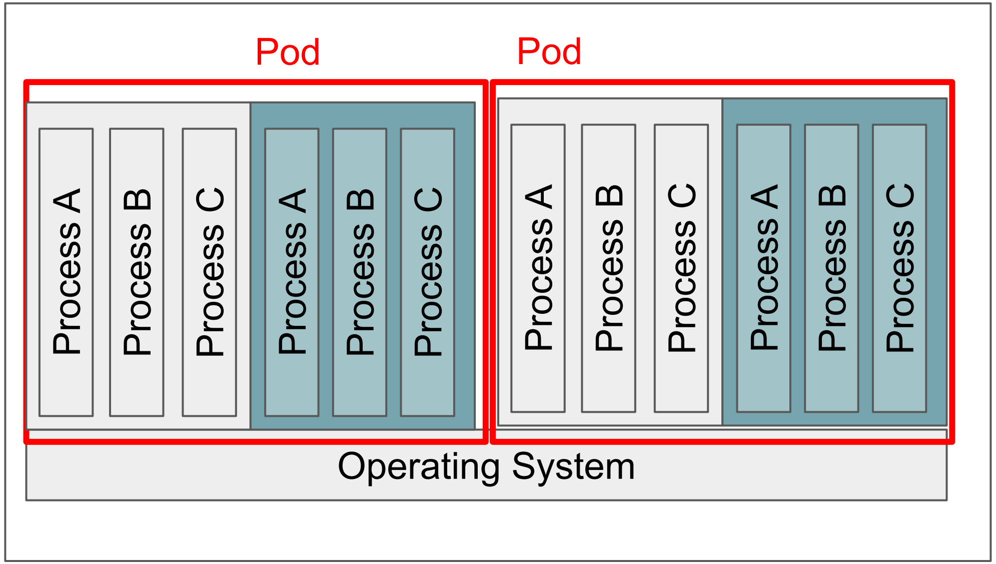
~~Computer~~ **Node**



Kubernetes - Execution

Do you manually create/delete pods? That would be a huge hassle.

~~Computer~~ **Node**



Pods are controlled usually via higher level abstractions (e.g., **Deployment, Jobs**)

Deployment

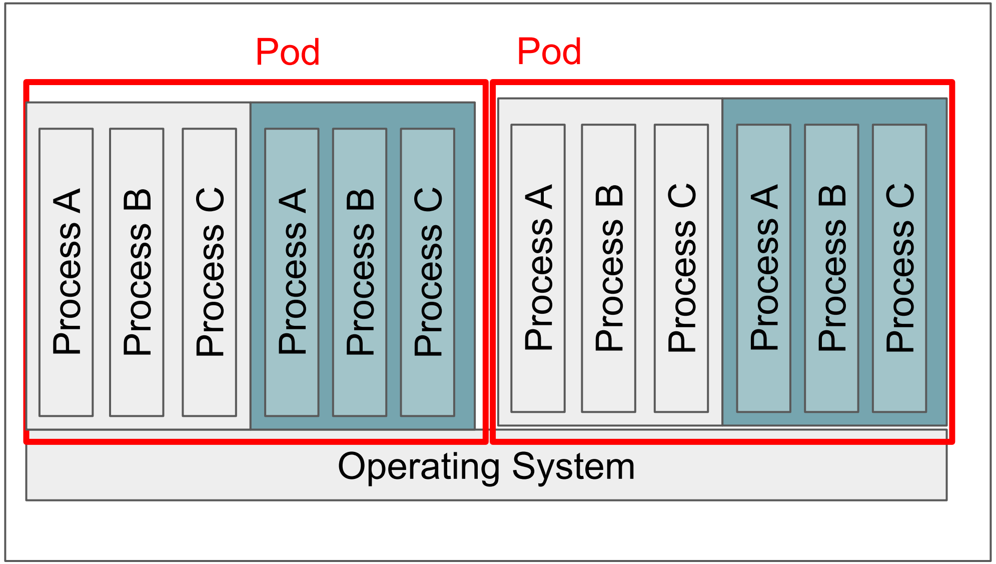
- declares the number of **pods** and what to execute in them
- Maintains that number of pods forever
- To scale up, create another deployment with new number of pods, similarly to scale down

Jobs
- Pod that runs a single time until end of execution and then is deleted
- Timeout

Kubernetes - Execution

What if I don't want to
even do manual scaling?

~~Computer~~ **Node**

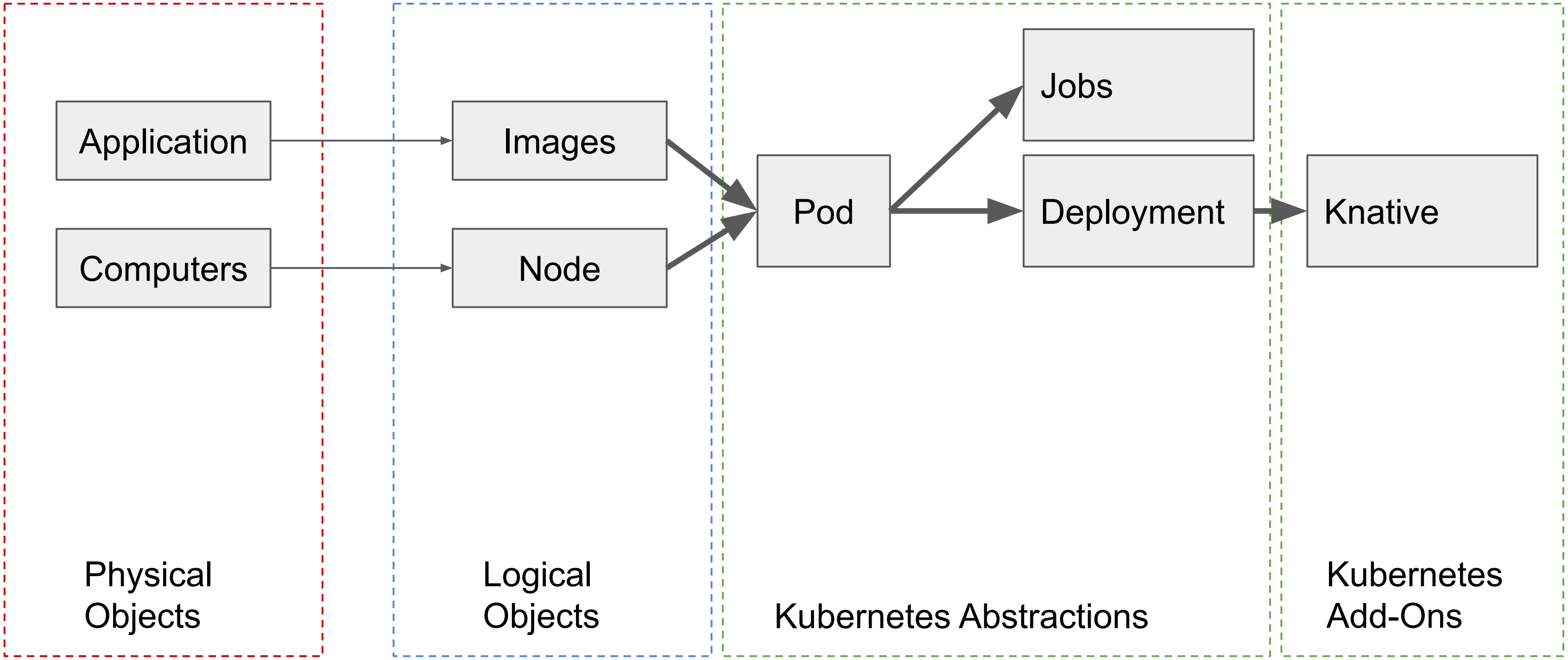


Answer: another layer of abstractions!
Knative!

Knative is a Kubernetes **Add-On**

- Add-on is a fancy word for a bunch of kubernetes abstractions packaged together
- Auto-scales the number of pods according to traffic/demand

Kubernetes - Execution



Kubernetes - Execution Recap

Recap:

- What is a pod?

- How many processes can run in a container?

- If you wanted multiple replicas of the same application, what would you do?

- If you had a web server and a database, how do you run them so that if a node fails, both the web server and database fail together? How would you run them if you wanted them to fail separately?

- What is the difference between a Kubernetes Add-On and Kubernetes abstraction (they are actually called resources, but let’s call them abstractions for generality)?

Kubernetes - Execution Recap

Recap:

- Who are the users of Kubernetes?
- What do developers of Kubernetes develop?

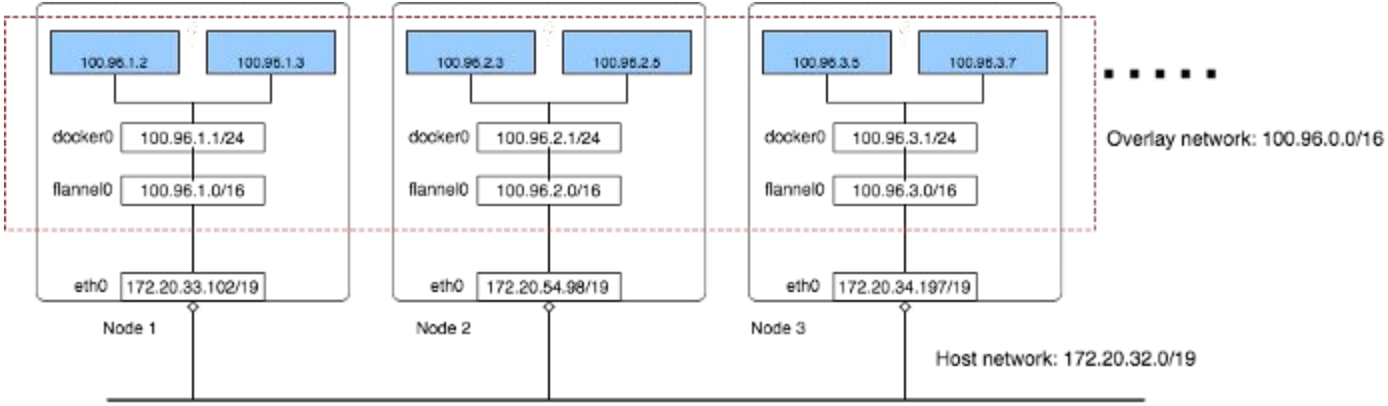
Kubernetes - Networking Model

But **nodes** are physical servers, they don’t have to be in the same network address space in the data center.

Kubernetes - Networking Model

But **nodes** are physical servers, they don't have to be in the same network address space in the data center.

- All containers in a pod are in the same network (as if local)
- All pods are in a flat address space (same subdomain)

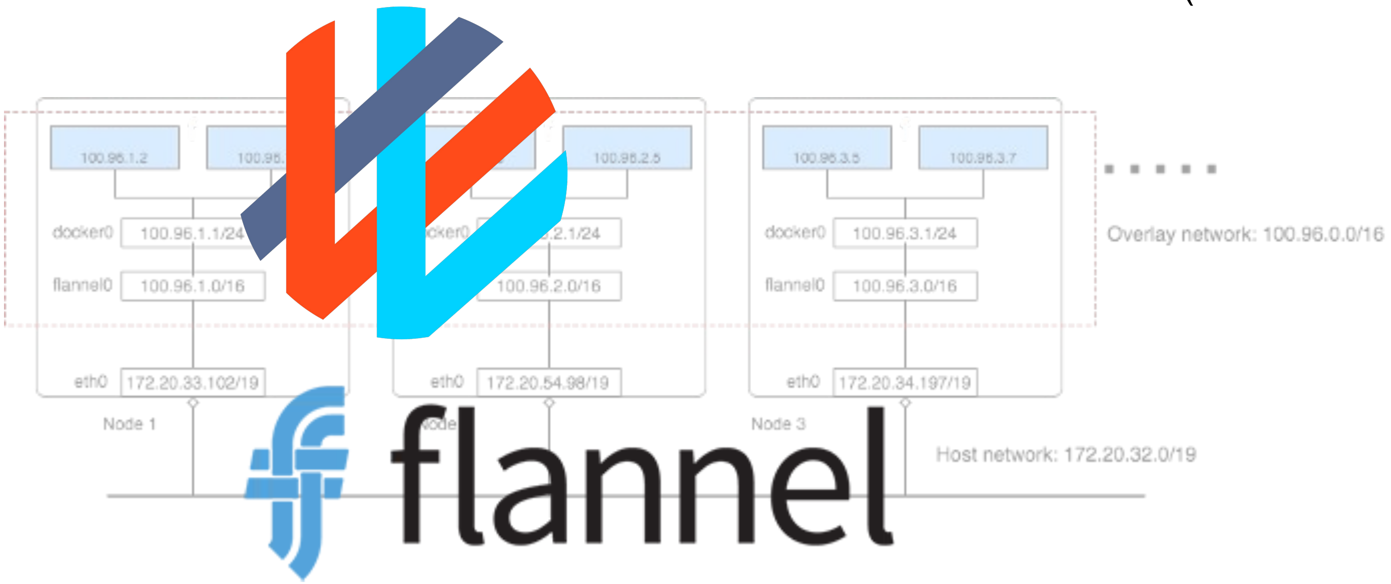


Kubernetes - Networking Model

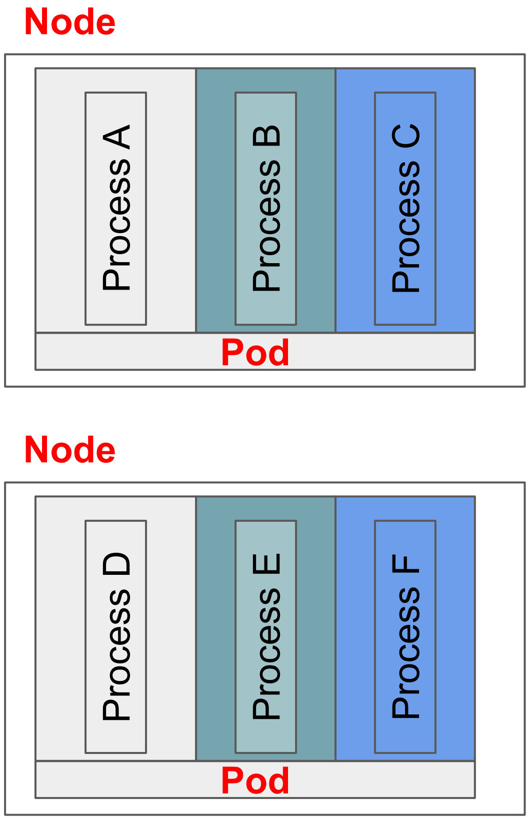
But **nodes** are physical servers, they don’t have to be in the same network address space in the data center.

Solution: overlay a logical network onto the physical network (e.g., Flannel, Weave, GCP, AWS, Azure, ….)

- All containers in a pod are in the same network (as if local)
- All pods are in a flat address space (same subdomain)

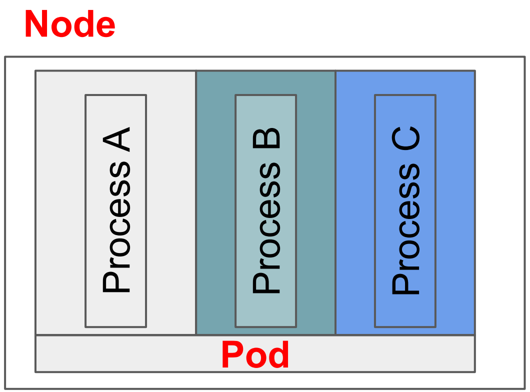


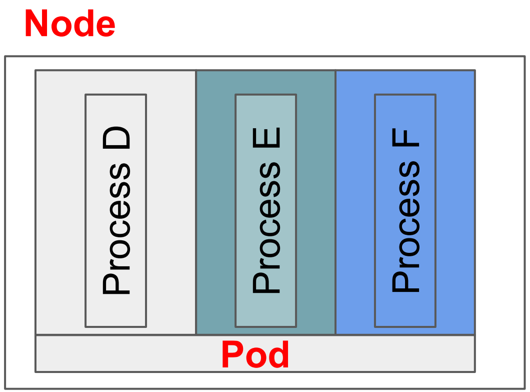
Kubernetes - Discovery



- All containers in a pod are in the same network (as if local)
- All pods are in a flat address space (same subdomain)
- Pods are dynamically allocated (initial position unknown) and **can move around** (e.g, replicas destroyed and re-created)

Kubernetes - Discovery

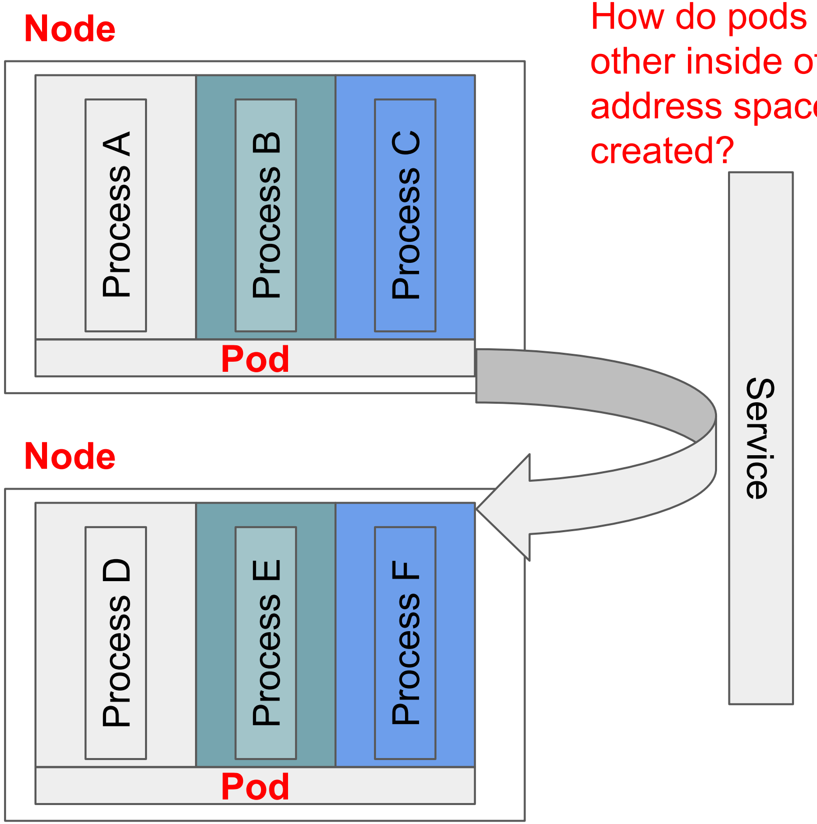




How do pods find each other inside of the flat address space we just created?

- All containers in a pod are in the same network (as if local)
- All pods are in a flat address space (same subdomain)
- Pods are dynamically allocated (initial position unknown) and can move around (e.g, replicas destroyed and re-created)

Kubernetes - Service

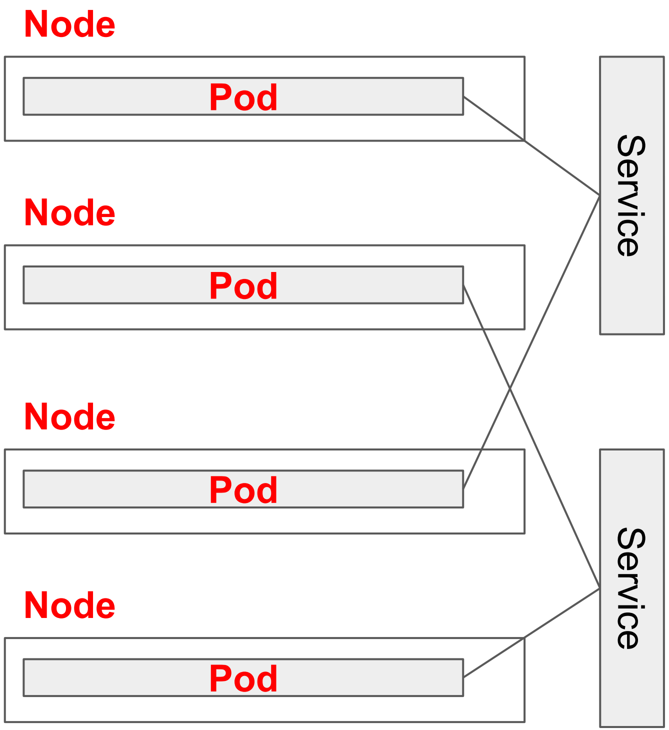


How do pods find each other inside of the flat address space we just created?

- All containers in a pod are in the same network (as if local)
- All pods are in a flat address space (same subdomain)
- Pods are dynamically allocated (initial position unknown) and can move around (e.g, replicas destroyed and re-created)
- **Service**: discovery (e.g., DNS), also inter-pod load-balancing

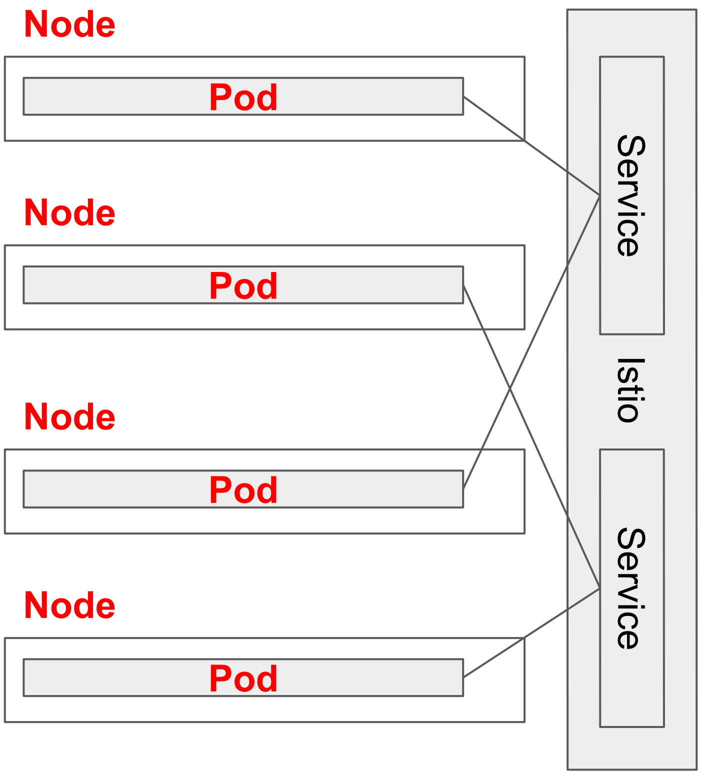
Kubernetes - Service Mesh

Microservices have a lot of services...how do I manage them?



Kubernetes - Service Mesh

Microservices have a lot of services...how do I manage them?

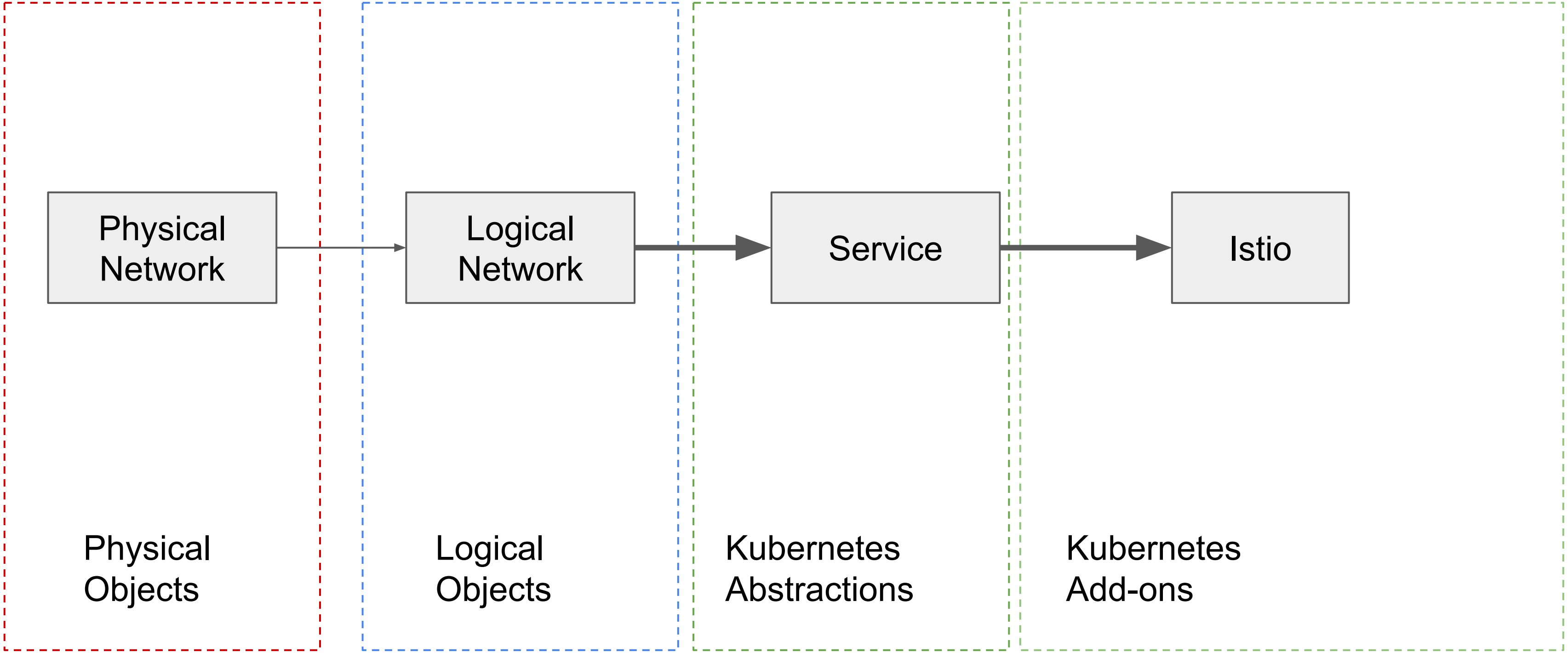


Answer: another layer of abstraction! Another add-on!

Istio: a service mesh manager

- **Control:** traffic splits (e.g., A/B testing)
- **Policies:** rate limiting between services
- **AAA**: authentication, authorization, etc.,
- **Observability**: tracing, metrics, logs

Kubernetes - Network Abstractions



Kubernetes - Network Abstractions Recap

Recap:

- What is needed to get physical servers on different racks to look like they are in the same subdomain? Note: each rack is a different subdomain

- Why would you not just deploy a cluster on a single server rack?

- How do your pods find each other?

Kubernetes - Network Abstractions Recap

Recap:

- What is needed to get physical servers on different racks to look like they are in the same subdomain? Note: each rack is a different subdomain

- Why would you not just deploy a cluster on a single server rack?

- How do your pods find each other?

- What is the difference between Istio and a Service?

Kubernetes - Persistence

- pods are ephemeral but some states need to persist

Ok but how do I store data persistently for the code I am running in a container in a pod? (e.g., pod get moved to another node, how do I keep the data I wrote to files on the previous node?

Kubernetes - Persistence

Ok but how do I store data persistently for the code I am running in a container in a pod? (e.g., pod get moved to another node, how do I keep the data I wrote to files on the previous node?

- pods are ephemeral but some states need to persist
    - execution (pods) are stateless, consistency semantics (multi-reader, multi-writer) are provided by different backing stores and restrictions (e.g., only one pod may mount a GCP persistent disk at a time)

Kubernetes - Persistence

Ok but how do I store data persistently for the code I am running in a container in a pod? (e.g., pod get moved to another node, how do I keep the data I wrote to files on the previous node?

- pods are ephemeral but some states need to persist
  - execution (pods) are stateless, consistency semantics (multi-reader, multi-writer) are provided by different backing stores and restrictions (e.g., only one pod may mount a GCP persistent disk at a time)
- **volumes**: mount points of **pod** during runtime
  - ephemeral (e.g., use file system on the **node**)
    - Each pod has its own isolated disk space
  - Persistent Disks (e.g., GCP Persistent Disk, AWS Elastic Block Store, Azure Disk, etc,...)
    - Mountable by a single pod at a time
  - Many more
- API calls to your favourite distributed data store (e.g., Spanner, S3, etc,...)

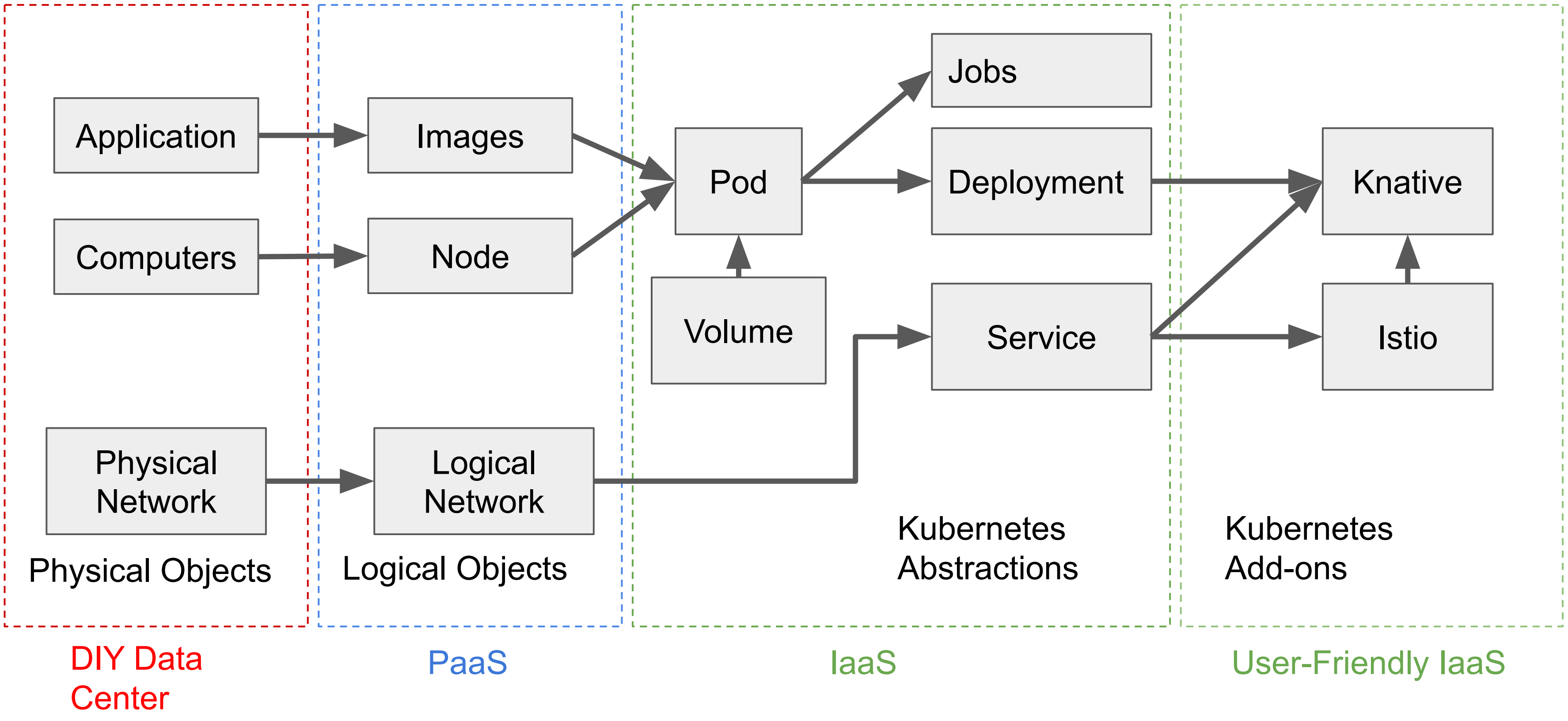
Kubernetes - Persistence Recap

Recap:

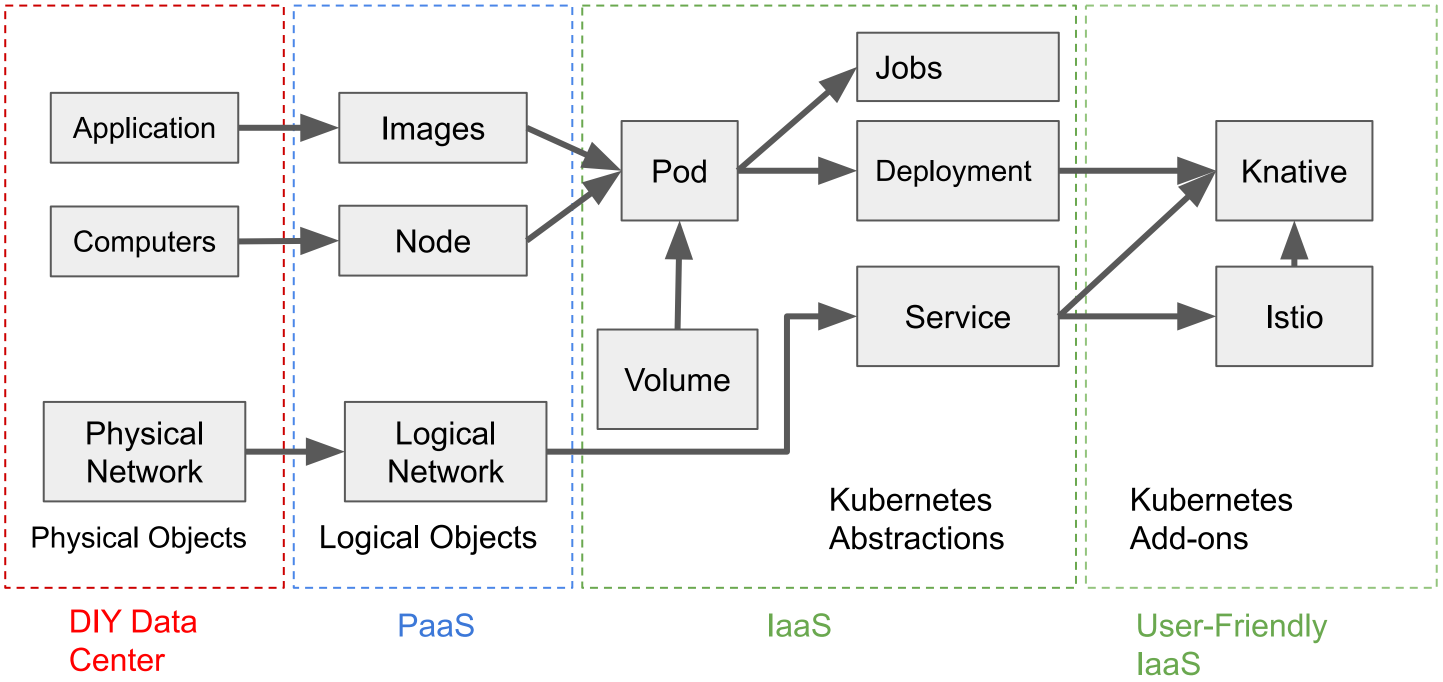
- I have an app that runs in a Kubernetes pod, and I want to store some user data. What are my options?

- What are the benefits of keeping “management” of persistent state out of Kubernetes execution?

Kubernetes - Execution + Network Abstractions = Dataplane

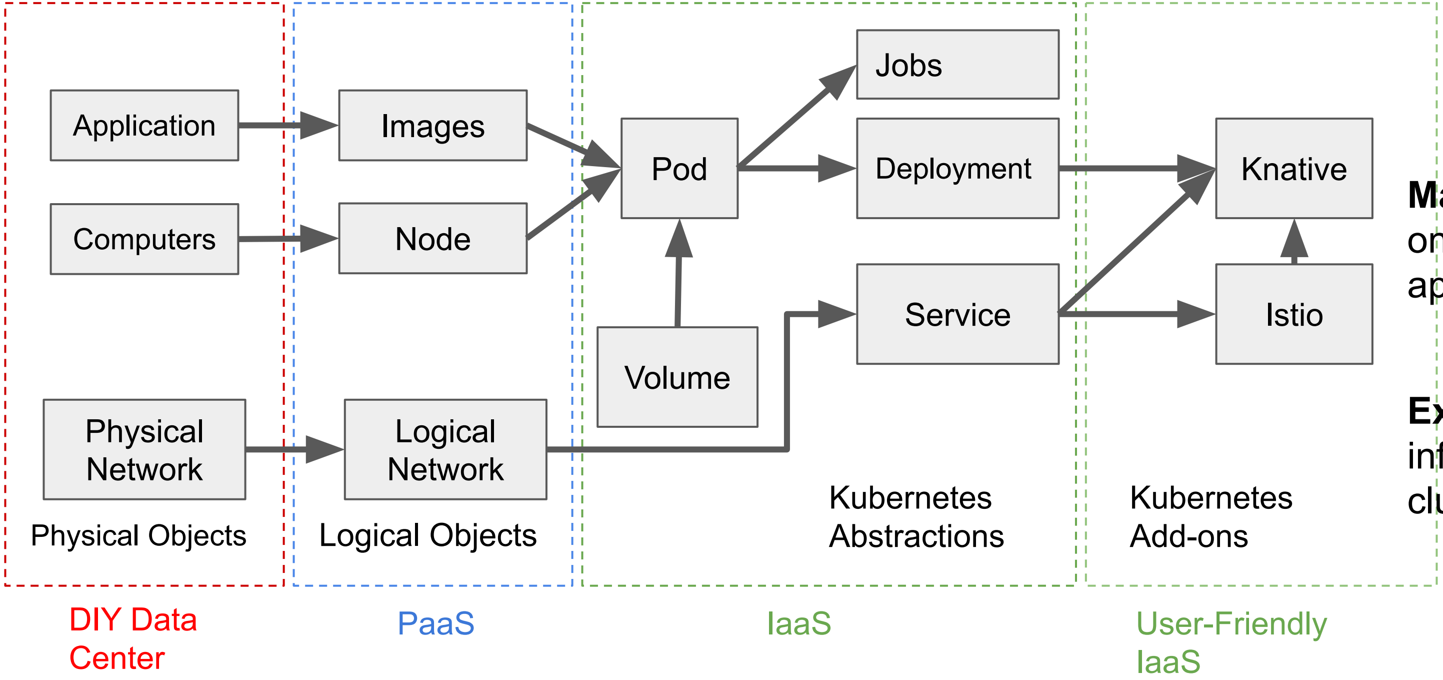


Kubernetes - Execution + Network Abstractions = Dataplane



Extensibility: build your own infrastructure (e.g., cluster-level abstractions)

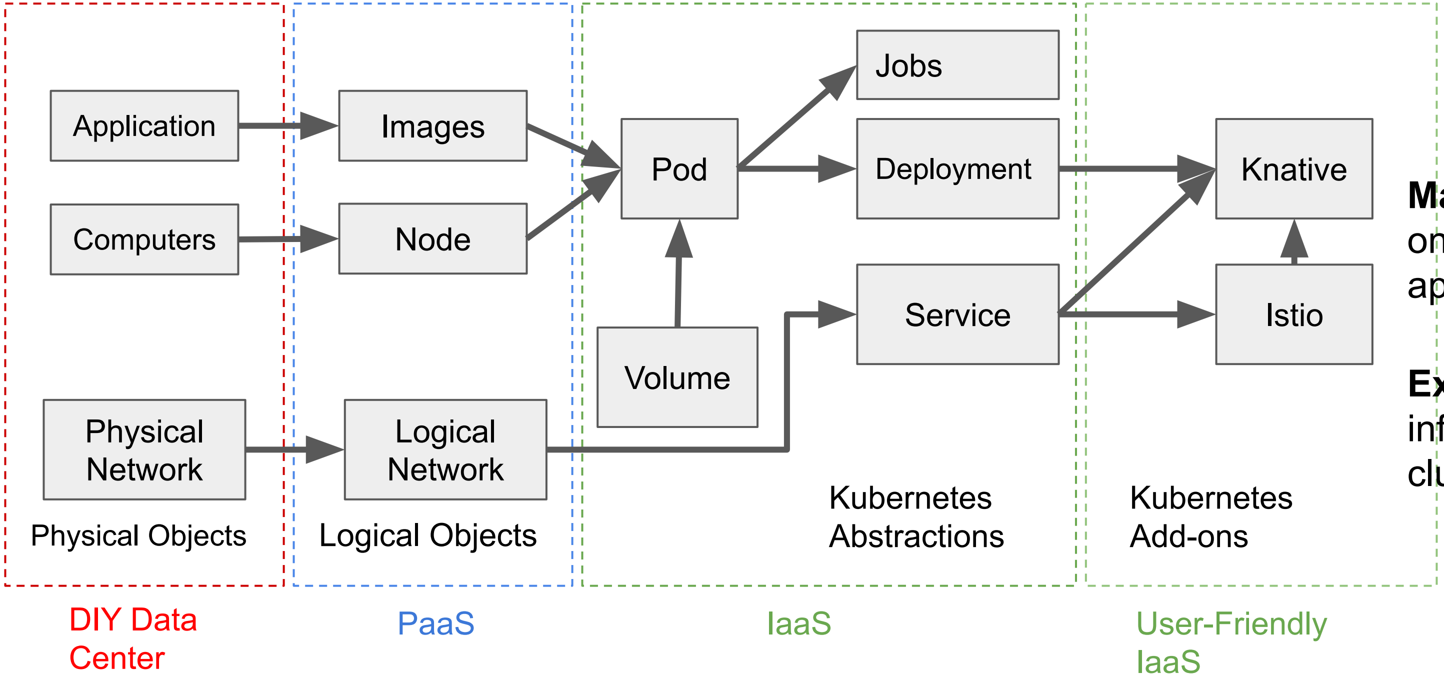
Kubernetes - Execution + Network Abstractions = Dataplane



Managed Services: users only need to care about applications

Extensibility: build your own infrastructure (e.g., cluster-level abstractions)

Kubernetes - Execution + Network Abstractions = Dataplane



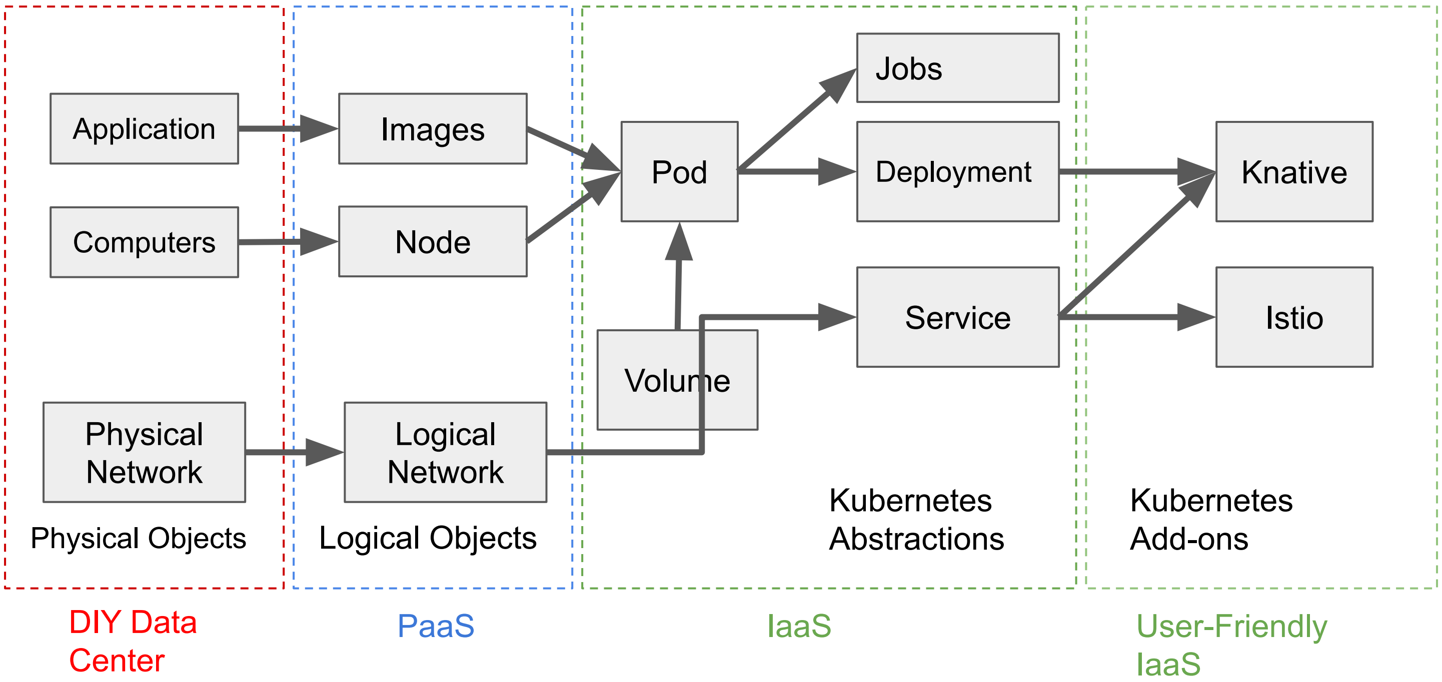
Managed Services: users only need to care about applications

Extensibility: build your own infrastructure (e.g., cluster-level abstractions)

Question: If you where an engineer developing a service, why would you use containers and Kubernetes to deploy your service, what value is it providing you and your organization?

Kubernetes - Management

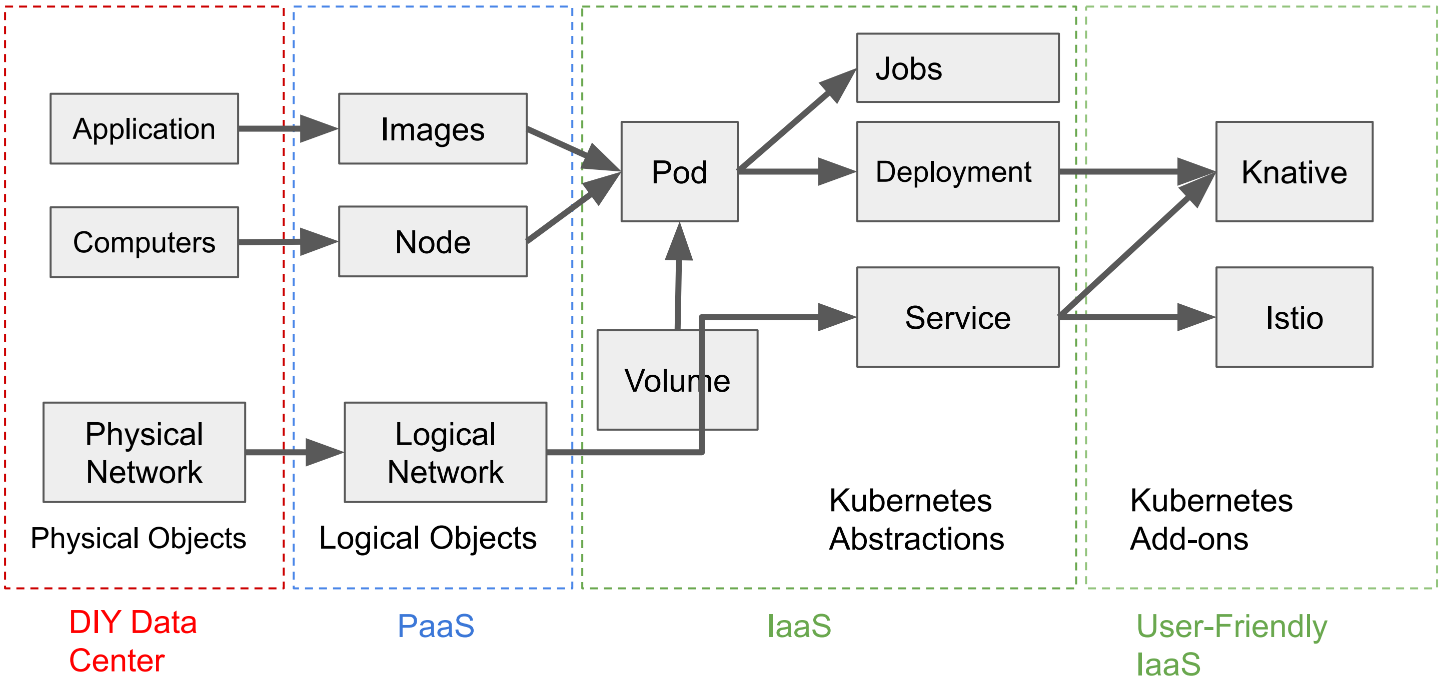
How is all this controlled and managed?



Kubernetes - Control Plane

How is all this controlled and managed?

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



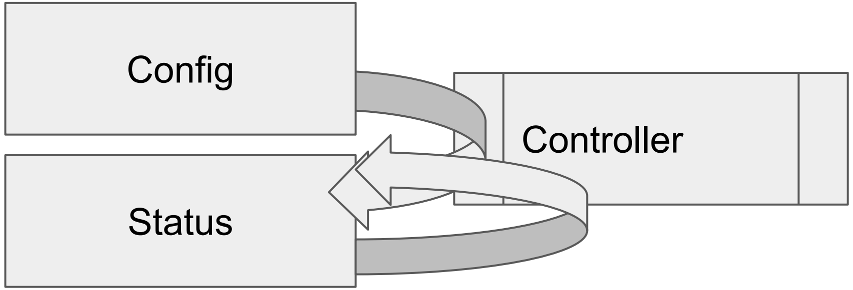
Kubernetes - Control Plane Controller Manager

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Where does the logic for the Kubernetes abstractions and add-ons all live?

Kubernetes - Control Plane Controller Manager

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



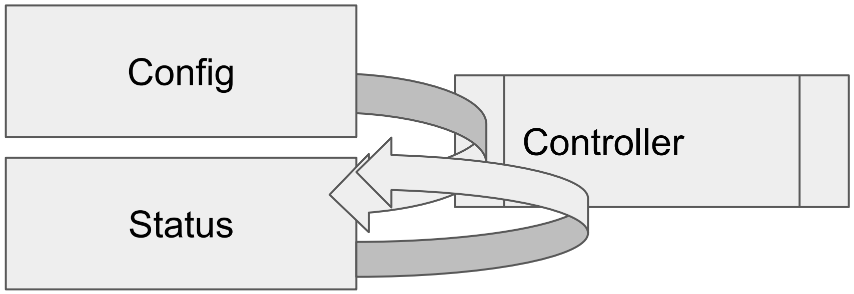
Where does the logic for the Kubernetes abstractions and add-ons all live?

A controller is:

- A non-terminating code loop that anneals state from desired to current
    - Eventually consistent
- Steps:
    - Look at config
    - Make changes to the cluster (e.g., create/delete pods based on the number configured in **Deployment**)
    - Write result to status

Kubernetes - Control Plane Controller Manager

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



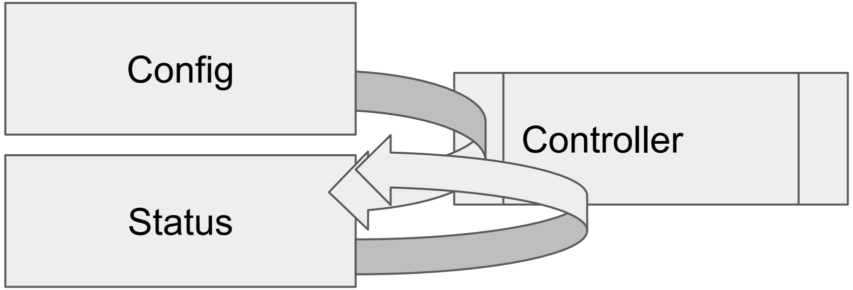
Where does the logic for the Kubernetes abstractions and add-ons all live?

Option 1: deployed as part of Kubernetes control plane

- Compiled together to create **controller manager**
- Naturally fault-tolerant with the control plane with leader and standby controller managers

Kubernetes - Control Plane Controller Manager

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



Where does the logic for the Kubernetes abstractions and add-ons all live?

Option 1: deployed as part of Kubernetes control plane

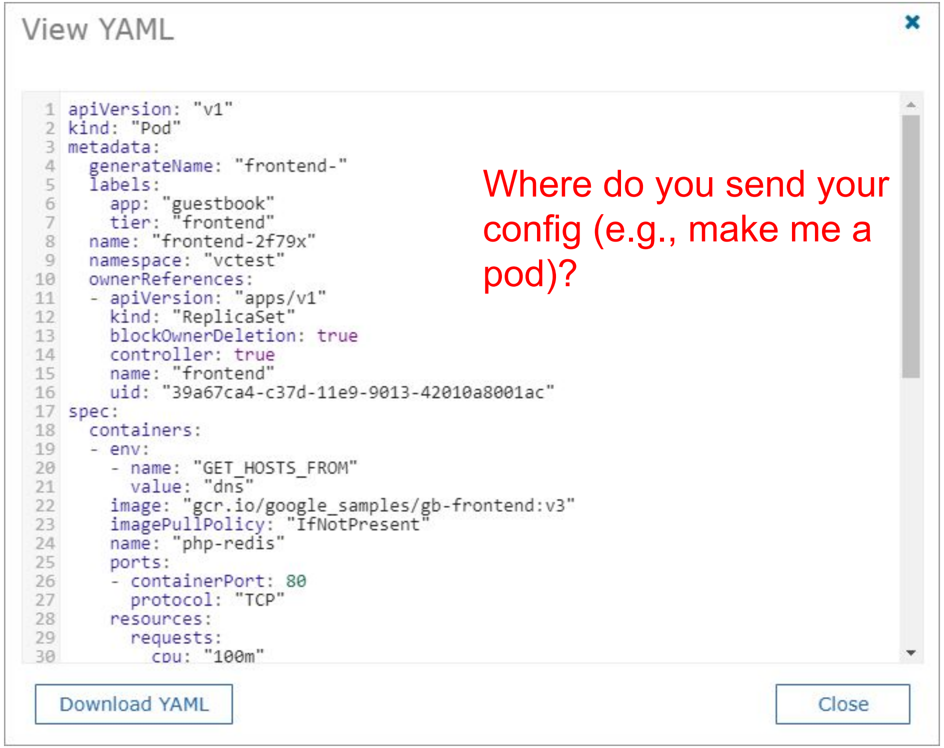
- Compiled together to create **controller manager**
- Naturally fault-tolerant with the control plane with leader and standby controller managers

Option 2: deployed as a pod (extensible)

- Developers self-manage leader election
- Keep only one replica that is managed by native Kubernetes applications (e.g., **Deployment**)

Kubernetes - Control Plane API Server

API Server | etcd | Controller Manager | Scheduler | Kubelet | Kubectl



Where do you send your config (e.g., make me a pod)?

Kubernetes - Control Plane API Server

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Declarative API

- **Resources**: Kubernetes abstractions such as **pod, service**, etc.,
- HTTP endpoints (e.g. apiextensions.k8s.io/v1)
- YAML

View YAML


```yaml
apiVersion: "v1"
kind: "Pod"
metadata:
  generateName: "frontend-"
  labels:
    app: "guestbook"
    tier: "frontend"
  name: "frontend-2f79x"
  namespace: "vctest"
  ownerReferences:
  - apiVersion: "apps/v1"
    kind: "ReplicaSet"
    blockOwnerDeletion: true
    controller: true
    name: "frontend"
    uid: "39a67ca4-c37d-11e9-9013-42010a8001ac"
spec:
  containers:
  - env:
    - name: "GET_HOSTS_FROM"
      value: "dns"
    image: "gcr.io/google_samples/gb-frontend:v3"
    imagePullPolicy: "IfNotPresent"
    name: "php-redis"
    ports:
    - containerPort: 80
      protocol: "TCP"
    resources:
      requests:
        cpu: "100m"
```


Where do you send your config (e.g., make me a pod)?

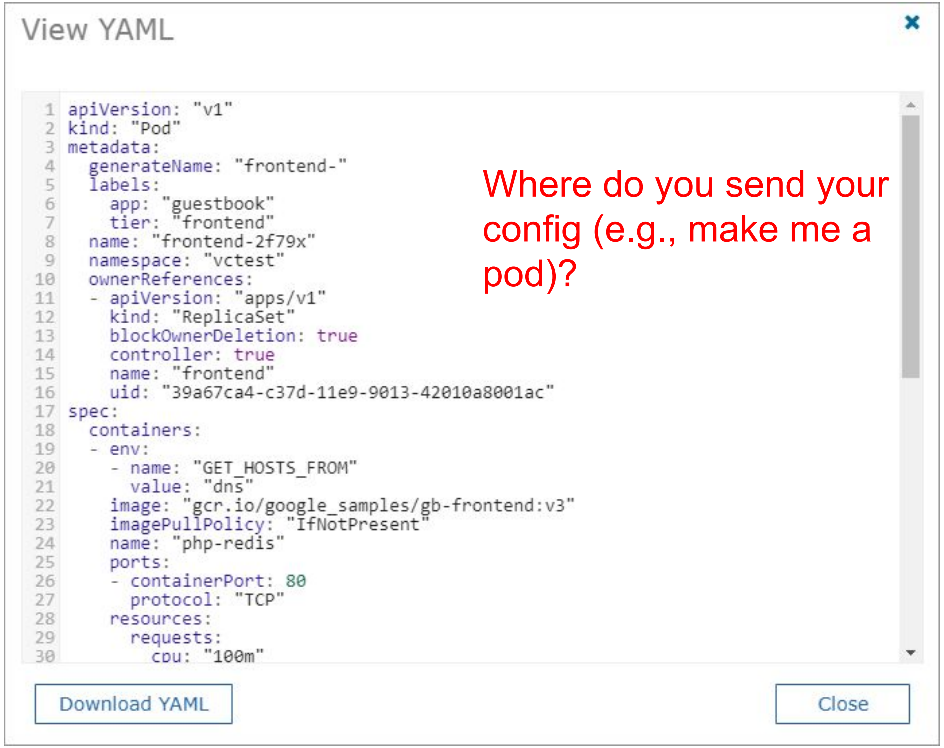
Download YAML　　Close

Kubernetes - Control Plane API Server

API Server | etcd | Controller Manager | Scheduler | Kubelet | Kubectl

Pass information through state

- eventually consistent
- Config (YAML) declares intent to **Controller**
- **Controller** polls intent, takes action and query result of its actions from cluster
- **Controller** writes the result of its actions to status
- Intent and result are made fault-tolerant via state, exist past the lifetime of controllers (e.g., not message passing)



Where do you send your config (e.g., make me a pod)?

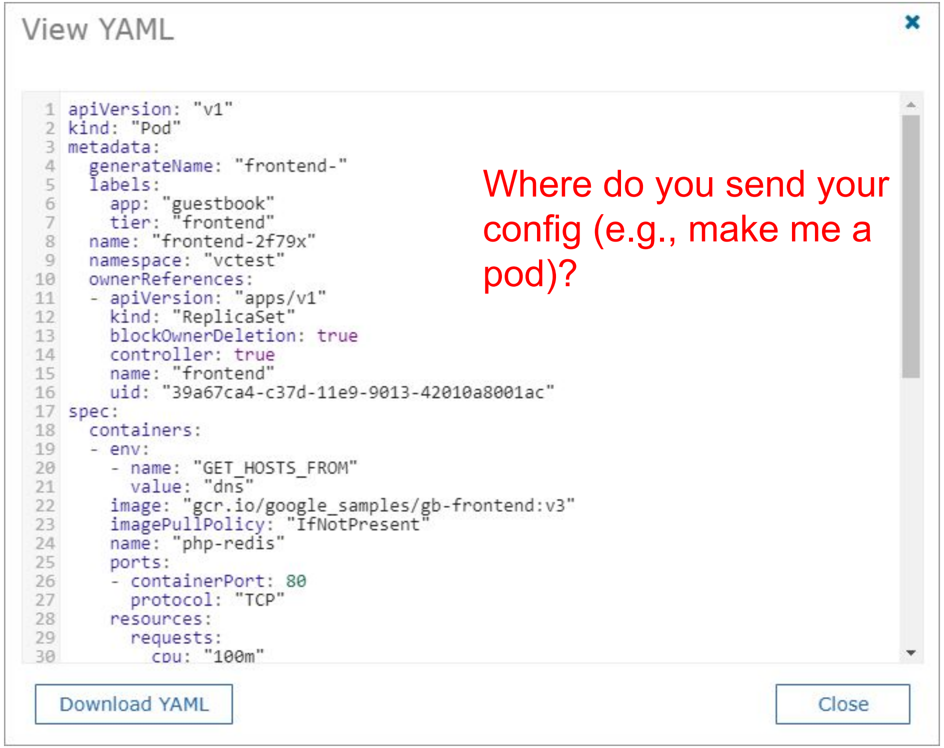
Kubernetes - Control Plane API Server

API Server | etcd | Controller Manager | Scheduler | Kubelet | Kubectl

Pass information through state

- eventually consistent
- Config (YAML) declares intent to **Controller**
- **Controller** react to intent, takes action and query result of its actions from cluster
- **Controller** writes the result of its actions to status
- Intent and result are made fault-tolerant via state, exist past the lifetime of controllers (e.g., not message passing)

Question: why does controller query the state of cluster, instead of just update the status based on the actions it took?


View YAML


```yaml
apiVersion: "v1"
kind: "Pod"
metadata:
  generateName: "frontend-"
  labels:
    app: "guestbook"
    tier: "frontend"
  name: "frontend-2f79x"
  namespace: "vctest"
  ownerReferences:
  - apiVersion: "apps/v1"
    kind: "ReplicaSet"
    blockOwnerDeletion: true
    controller: true
    name: "frontend"
    uid: "39a67ca4-c37d-11e9-9013-42010a8001ac"
spec:
  containers:
  - env:
    - name: "GET_HOSTS_FROM"
      value: "dns"
    image: "gcr.io/google_samples/gb-frontend:v3"
    imagePullPolicy: "IfNotPresent"
    name: "php-redis"
    ports:
    - containerPort: 80
      protocol: "TCP"
    resources:
      requests:
        cpu: "100m"
```


Where do you send your config (e.g., make me a pod)?

Kubernetes - Control Plane API Server

API Server | etcd | Controller Manager | Scheduler | Kubelet | Kubectl

Pass information through state

- eventually consistent
- Config (YAML) declares intent to **Controller**
- **Controller** react to intent, takes action and query result of its actions from cluster
- **Controller** writes the result of its actions to status
- Intent and result are made fault-tolerant via state, exist past the lifetime of controllers (e.g., not message passing)

Question: what are the down-side of message passing? (e.g., send config to controller directly and getting status back)

View YAML


```yaml
apiVersion: "v1"
kind: "Pod"
metadata:
  generateName: "frontend-"
  labels:
    app: "guestbook"
    tier: "frontend"
  name: "frontend-2f79x"
  namespace: "vctest"
  ownerReferences:
  - apiVersion: "apps/v1"
    kind: "ReplicaSet"
    blockOwnerDeletion: true
    controller: true
    name: "frontend"
    uid: "39a67ca4-c37d-11e9-9013-42010a8001ac"
spec:
  containers:
  - env:
    - name: "GET_HOSTS_FROM"
      value: "dns"
    image: "gcr.io/google_samples/gb-frontend:v3"
    imagePullPolicy: "IfNotPresent"
    name: "php-redis"
    ports:
    - containerPort: 80
      protocol: "TCP"
    resources:
      requests:
        cpu: "100m"
```

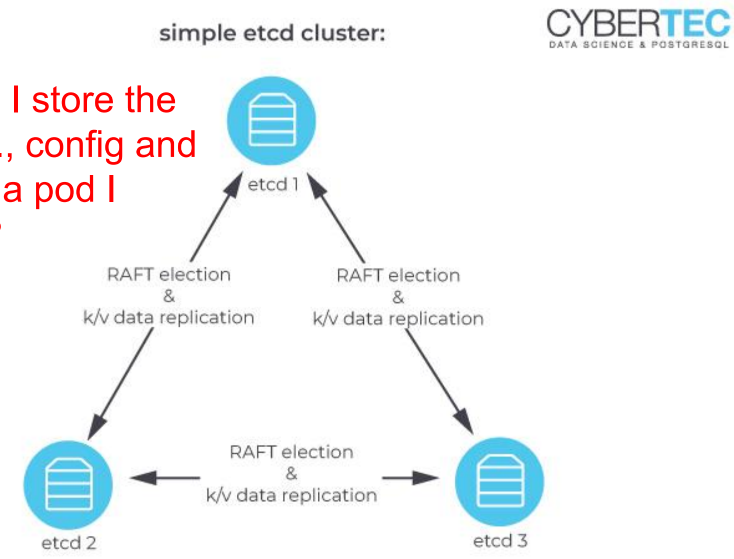

Download YAML | Close

Where do you send your config (e.g., make me a pod)?

Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

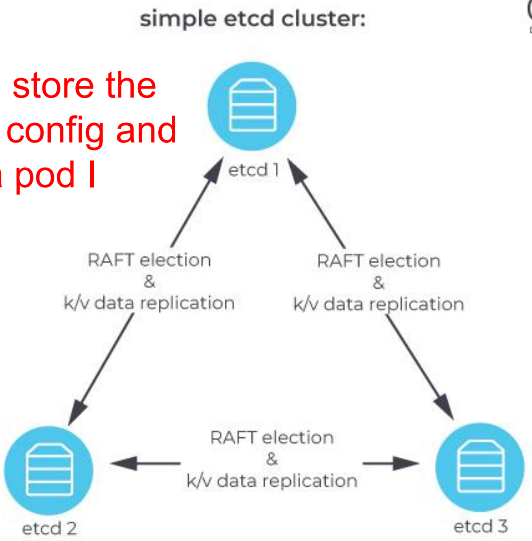
Where do I store the state (e.g., config and status) of a pod I declared?



Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Where do I store the state (e.g., config and status) of a pod I declared?

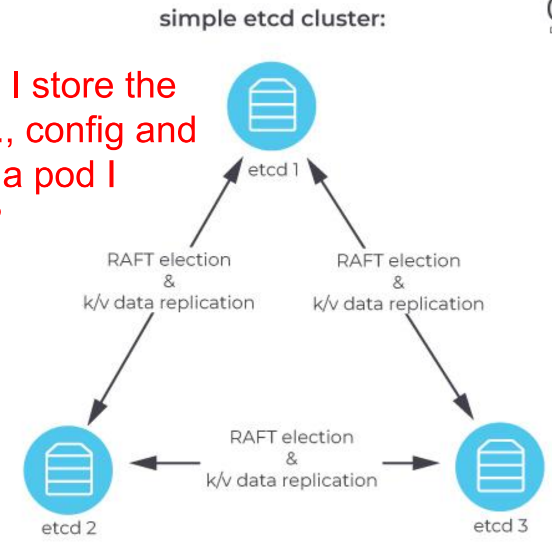


Etcd is a distributed key-value store for cluster states (e.g., API configs, metadata)

Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Where do I store the state (e.g., config and status) of a pod I declared?

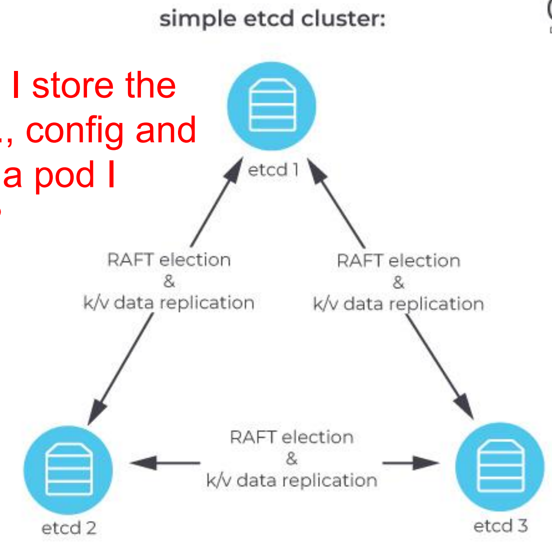


It does:

- state replication
  - leader writes to logs which are replicated to non-leader nodes

Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



CYBERTEC DATA SCIENCE & POSTGRESQL

Where do I store the state (e.g., config and status) of a pod I declared?

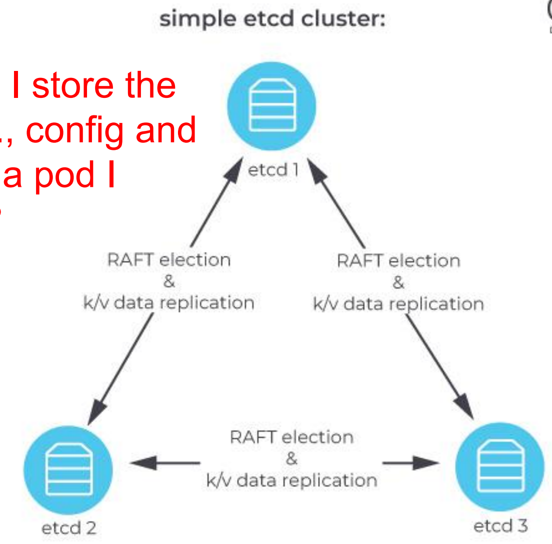
It does:

- state replication
  - leader writes to logs which are replicated to non-leader nodes
- leader election
  - based on Raft (a version of Paxos where there is a trusted leader)

Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Where do I store the state (e.g., config and status) of a pod I declared?



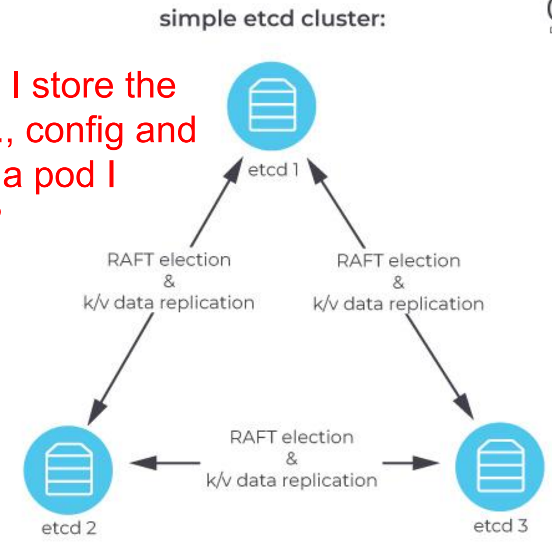
It does:

- state replication
  - leader writes to logs which are replicated to non-leader nodes
- leader election
  - based on Raft (a version of Paxos where there is a trusted leader)
- distributed Locks
  - leader handles lease expiration

Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Where do I store the state (e.g., config and status) of a pod I declared?



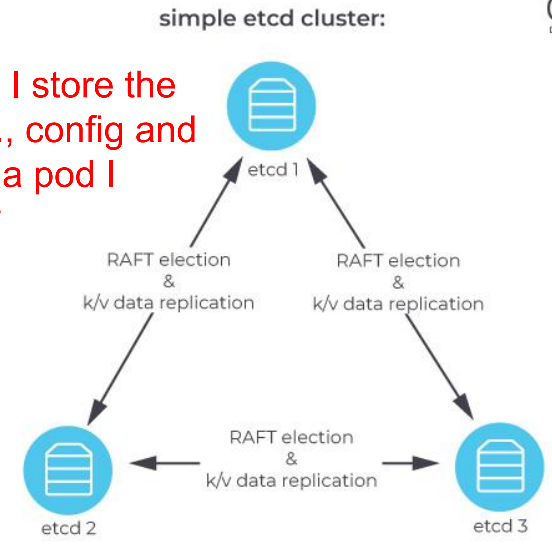
It does:

- state replication
  - leader writes to logs which are replicated to non-leader nodes
- leader election
  - based on Raft (a version of Paxos where there is a trusted leader)
- distributed Locks
  - leader handles lease expiration
- consistency
  - no transactions (not ACID), but have a “transaction abstraction” for compare-and-swap
  - linearizable read
  - Versioned writes with compaction

Kubernetes - Control Plane State

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Where do I store the state (e.g., config and status) of a pod I declared?



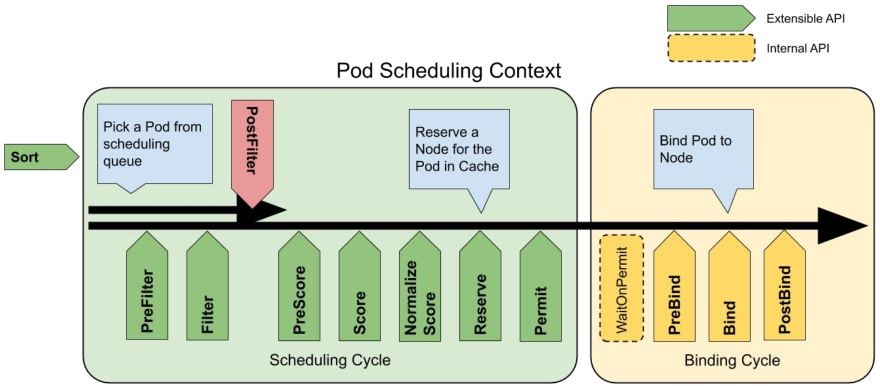
Time to tie everything together.

Questions/Recap

- Why is etcd needed at all? (e.g., how does etcd, controllers and API server all fit together)

Kubernetes - Control Plane Scheduler

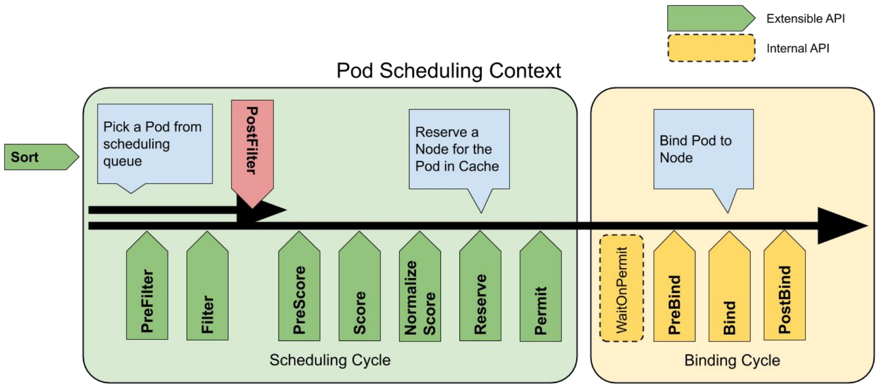
API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



How do I do resource control (e.g., map pods to nodes)?

Kubernetes - Control Plane Scheduler

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl



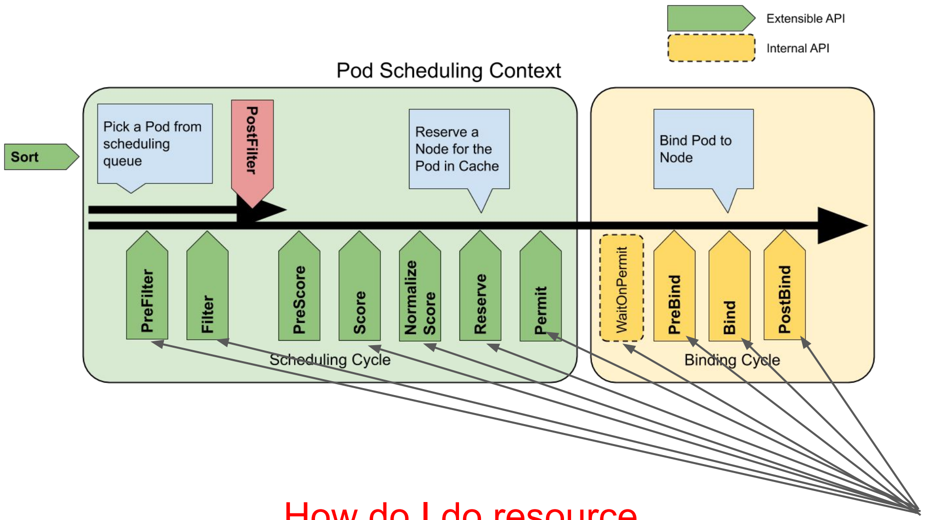
Scheduler calculates a score for which node to run a pod on based on the pod config which contains:

- affinity, anti-affinity
- resource requirements and availability
- soft tolerations and hard constraints (e.g., run only on nodes with label GPU)
- evictability

How do I do resource control (e.g., map pods to nodes)?

Kubernetes - Control Plane Scheduler

API Server | etcd | Controller Manager | Scheduler | Kubelet | Kubectl



Scheduler calculates a score for which node to run a pod on based on the pod config which contains:

- affinity, anti-affinity
- resource requirements and availability
- soft tolerations and hard constraints (e.g., run only on nodes with label GPU)
- evictability

Extremely extensible, can interpose at any of the points in the scoring system

How do I do resource control (e.g., map pods to nodes)?

Kubernetes - Control Plane Kubelet/Kubectl

API Server	etcd	Controller Manager	Scheduler	Kubelet	Kubectl

Kubelet

- Daemon running on the node that executes commands by the Kubernetes control plane (e.g., start/evict a pod on the node)

Kubectl

- Command-line interface for the Kubernetes cluster (talk to API server)
- What you use to interface with your Kubernetes Cluster

Kubernetes - Control Plane Data Model

Multi-Readers to etcd

- Reads can be linearizable (go to leader etcd node) or serializable (go to any of the replica nodes in etcd)

Kubernetes - Control Plane Data Model

Multi-Readers to etcd

- Reads can be linearizable (go to leader etcd node) or serializable (go to any of the replica nodes in etcd)

Multi-Writers to etcd

- [Option 1] don’t really care - everything is eventually consistent by observing the state of the cluster and then trying to get the cluster there
  - Coincident writes are merge/add/delete operation based on data type (single values vs. lists/maps) and last write wins
  - Transient inconsistencies are okay (e.g., 3 replicas, but might overshoot or undershoot temporarily)
  - Controllers fact-check with the actual world instead of state in etcd, what-you-see-is-eventually-what-you-get

Kubernetes - Control Plane Data Model

Multi-Readers

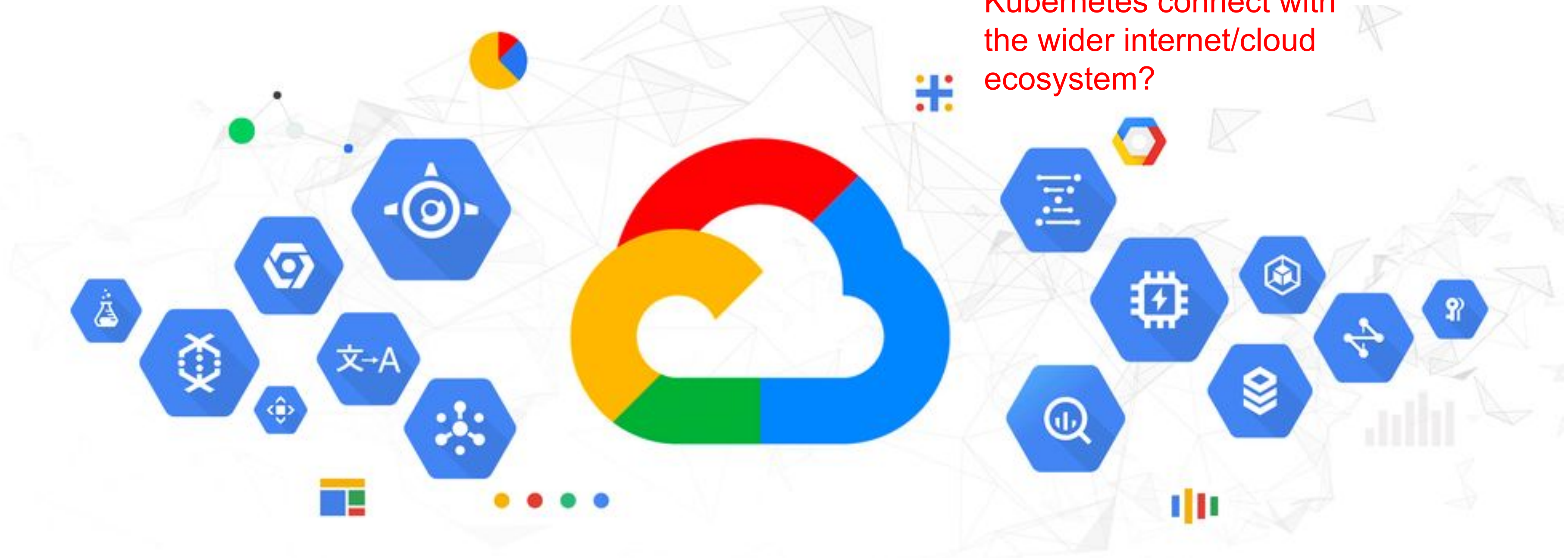
- Reads can be linearizable (go to leader etcd node) or serializable (go to any of the replica nodes in etcd)

Multi-Writers

- [Option 1] don’t really care - everything is eventually consistent by observing the state of the cluster and then trying to get the cluster there
    - Coincident writes are merge/add/delete operation based on data type (single values vs. lists/maps) and last write wins
    - Transient inconsistencies are okay (e.g., 3 replicas, but might overshoot or undershoot temporarily)
    - Controllers fact-check with the actual world instead of state in etcd, what-you-see-is-eventually-what-you-get
- [Option 2] use etcd in the control plane for serialization with distributed locks

Kubernetes - Integration

How does what I have in Kubernetes connect with the wider internet/cloud ecosystem?

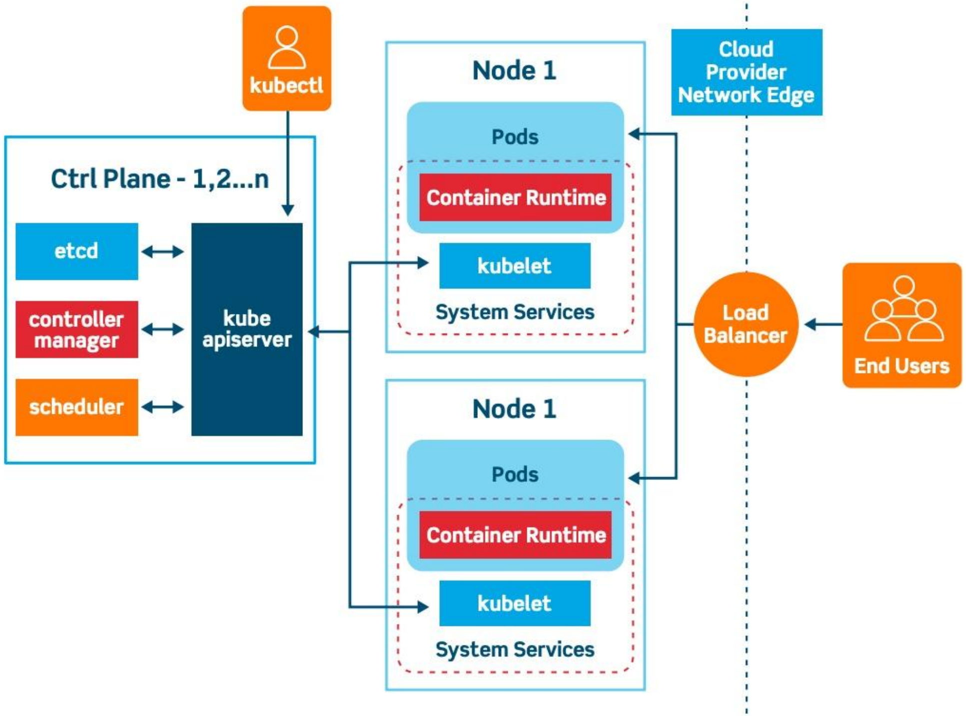


Load balancer (Internet), Security (IAM), Storage (Google Cloud Storage, Spanner, Google Container Registry), Events (Pub/Sub), Graph (Cloud Build, Google Dataflow), AI (Google Cloud AI)

Kubernetes - Summary

User Apps	Kubernetes API apps (e.g., controllers, sidecars, middleboxes)
Kubernetes APIs (e.g., pods, services, deployments, etc.,...)	Kubernetes Add-ons (Istio, Knative)
Kubernetes Control Plane (scheduler, API server, controller-manager, etcd)	
Logical network (Flannel, Weave, bespoke cloud provider implementation)	
Physical servers and network	

Kubernetes - Architecture



- Abstractions: Pods, Services, Ingress, Deployments, Volumes…
- Add-ons: Istio, Knative...
- Network: Weave, Flannel…
- Control Plane: scheduler, api-server, controller-manager, etcd...

Kubernetes - Interesting Problems

Dependency

- Controllers are all eventually consistent and order agnostic (ideall), but some abstractions have dependencies as outcome of implementation
- Worse, administration of Kubernetes clusters are usually split (between user and cloud provider, neither can be sure what the other has installed)

Configuration

- Thousands of lines of YAML with relationship defined by string **labels**

Debuggability

- Logs are spread out over multiple nodes, something goes wrong, how do you find out what went wrong?
- Distributed system debugging

Efficiency

- Essentially you build applications (containers) that run in cluster wide applications (also defined/built by you) that run on Kubernetes framework (lightweight, but still a cost to it)