Container Orchestration (with Kubernetes)
Marketing

Peter Chen
About Me

- 2014-2016: grad student at UBC (Ivan was my supervisor)
- 2016-2019: Arista Networks
- 2019-2021: Google Cloud (Knative)
- 2021-Now: Google Research (NLP and Speech)
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?

- ~2.9K 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, Microsoft, 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,...
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.

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

**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

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

Nice things about containers, they are:

- **Lightweight**: fast, very little overhead
- **Isolation**: executable package of software with its own code, runtime, libraries and settings
- **Portable**: compiled into an image
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 - Architecture

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

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

Too many big words that have too little meaning
Kubernetes - Execution

Computer

Operating System

Kubernetes - Execution
Kubernetes - Execution
Kubernetes - Execution

Computer Node

Pod

Pod

Operating System
Kubernetes - Execution

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

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

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

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

Computer Node

Operating System
Kubernetes - Execution

Physical Objects:
- Application
- Computers

Logical Objects:
- Images
- Node

Kubernetes Abstractions:
- Pod
- Deployment
- Jobs

Kubernetes Add-Ons:
- Knative
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)?
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

- 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

How do pods find each other inside of the flat address space we just created?
Microservices have a lot of services...how do I manage them?
Kubernetes - Service Mesh

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

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

- 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)

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

- 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,...)

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 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

Application

Images

Pod

Volume

Jobs

Deployment

Service

Knative

Istio

Computers

Node

Physical Objects

Physical Network

Logical Network

Logical Objects

Kubernetes Abstractions

Kubernetes Add-ons

IaaS

PaaS

DIY Data Center

User-Friendly IaaS
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)

DIY Data Center  PaaS  IaaS  User-Friendly IaaS
**Question:** If you were 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?

Application
Images
Node
Pod
Volume
Jobs
Deployment
Service
Knative
Istio

Physical Network
Logical Network

Physical Objects
Logical Objects

Kubernetes Abstractions

Kubernetes Add-ons

DIY Data Center
PaaS
IaaS
User-Friendly IaaS
Kubernetes - Control Plane

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

How is all this controlled and managed?

Application  |  Images  |  Pod  |  Jobs  |  Knative  |  Kubernetes
Computers  |  Node  |  Volume  |  Deployment  |  Istio  |  Add-ons

Logical Network  |  Physical Network

Physical Objects  |  Logical Objects

DIY Data Center  |  PaaS  |  IaaS

Physical Objects  |  Logical Network

IaaS  |  User-Friendly IaaS
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

Where does the logic for the Kubernetes abstractions and add-ons all live?
Kubernetes - Control Plane Controller Manager

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

Where does the logic for the Kubernetes abstractions and add-ons all live?
Kubernetes - Control Plane Controller Manager

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)

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

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

Config  

Controller  

Status
Kubernetes - Control Plane API Server

Where do you send your config (e.g., make me a pod)?
Kubernetes - Control Plane API Server

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

Where do you send your config (e.g., make me a pod)?
Kubernetes - Control Plane API Server

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

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)
Kubernetes - Control Plane API Server

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?
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)

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

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

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)

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?

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

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

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

Questions/Recap

- Why is etcd needed at all? (e.g., how does etcd, controllers and API server all fit together)
How do I do resource control (e.g., map pods to nodes)?
Kubernetes - Control Plane Scheduler

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

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

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

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)

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

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

Debggability

- 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)
Acknowledgement

Thanks to Jon Donovan for helping me edit this