

JupyterCon - Machine Learning at Scale with Kubernetes

Aug 23rd, 2018

Chris Cho, Fei Xue, Tyler Erickson

Google Cloud



Kubeflow



Contents

1. Containers
2. Kubernetes
3. Google Kubernetes Engine
4. Kubeflow - Overview
5. Kubeflow - Components
6. Kubeflow - Architecture
7. ksonnet
8. Kubeflow - Tutorial

Appendix- Tensor2Tensor

Agenda

Start Time	Section Total	Duration (Min)	Type	Topic area
9:00 AM		0:30	Presentation	Welcome + Introduction + Setup
9:30 AM		0:45	Exercise	GKE quest lab 1- Introduction to docker
10:15 AM		0:05	Debrief	
10:20 AM		0:30	Presentation	Kubernetes and GKE
10:50 AM		1:00	Exercise	GKE quest lab 2 - Hello Node Kubernetes
11:50 AM		0:05	Debrief	
11:55 AM		0:30	Presentation	Orchestrating the Cloud with GKE
12:25 PM	5:25	1:00	Lunch	
1:25 PM		1:00	Exercise	GKE quest lab 3 - Orchestrating the Cloud with GKE
1:25 PM		0:30	Presentation	Kubeflow introduction
1:55 PM		1:00	Exercise	Kubeflow 1 - Intro to Kubeflow
2:55 PM		0:05	Debrief	
3:00 PM		0:30	Presentation	Kubeflow - Components, Architecture, Short tutorial
3:00 PM		1:30	Exercise	Kubeflow 2 - E2E
4:30 PM	3:35	Buffer		

Expectations

- Walk away with better understanding of...
 - Containers/Docker
 - Kubernetes
 - Orchestrating Kubernetes
 - Kubeflow
- Try out Kubeflow when you get home :)

Logistics

- Access you will need for the labs
 - 8 Qwiklab credits - Pick it up in the back
 - 2 GCP test accounts - Handed out at the end
- Sign up on <https://qwiklabs.com>
- Enroll in quest <https://qwiklabs.com/quests/29>

Containers

Agenda

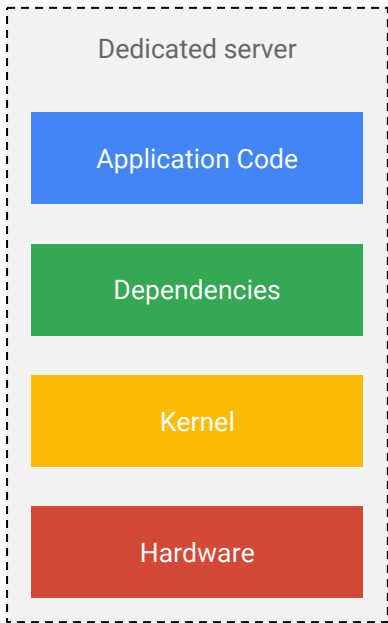
What are containers?

What is Docker?

Why containers?

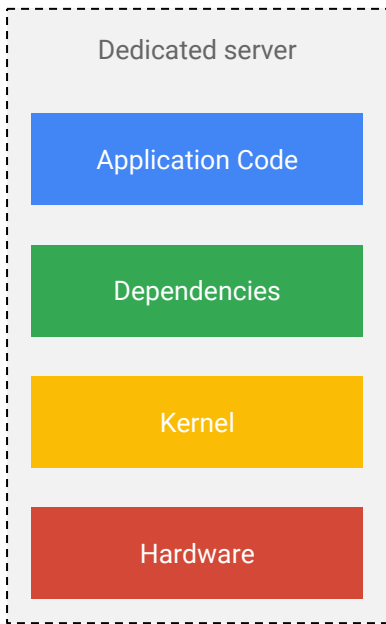
Lab, Container Fundamentals

A Machine

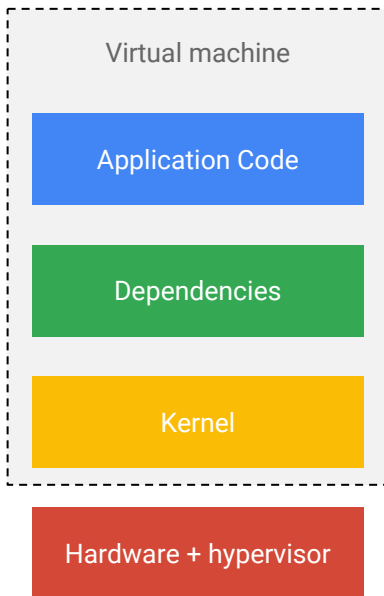


Deployment ~months
Low utilization
Not portable

A Virtual Machine

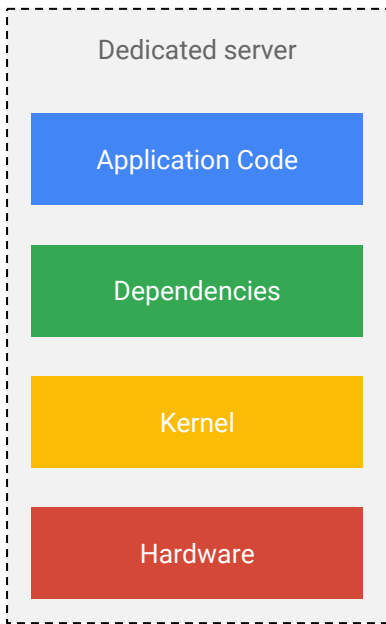


Deployment ~months
Low utilization
Not portable

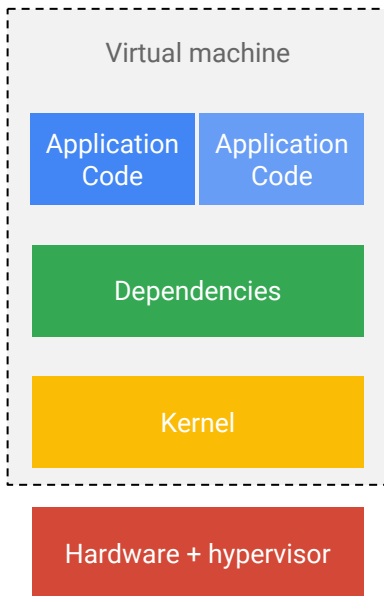


Deployment ~days (mins)
Improved utilization
Hypervisor specific

Many Virtual Machines

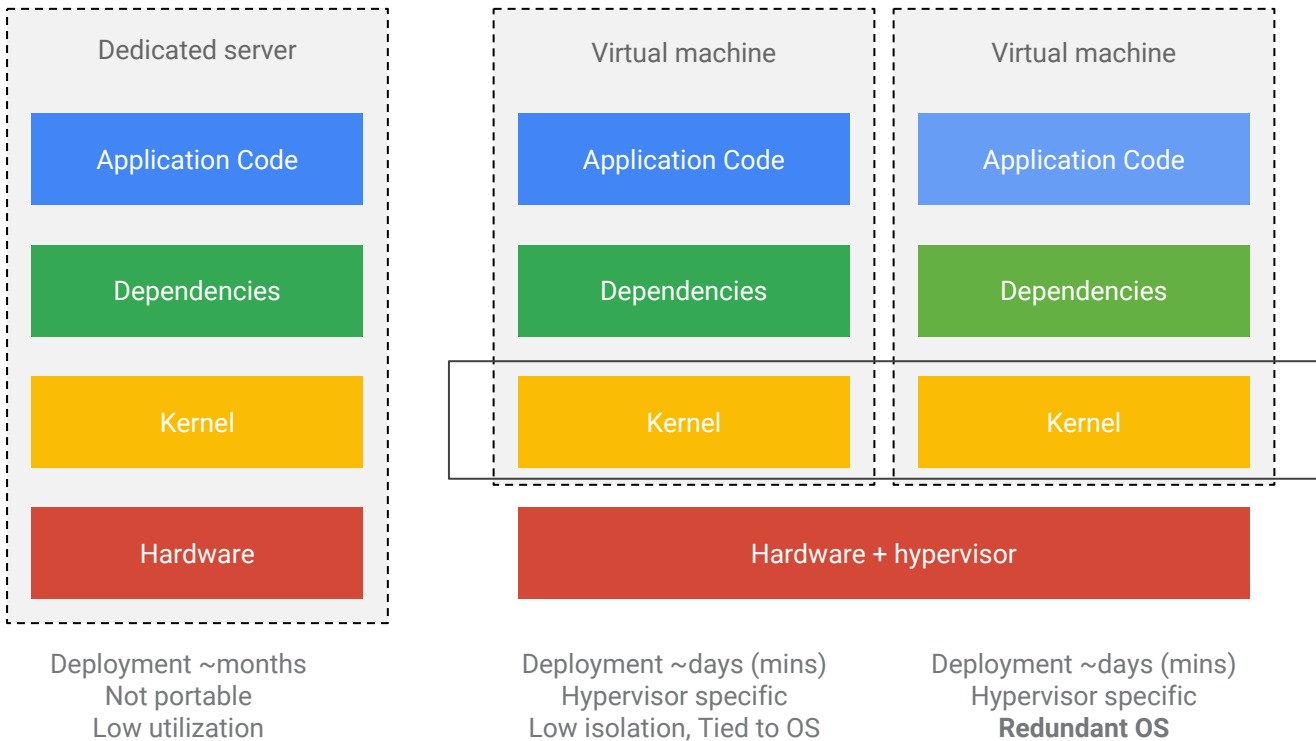


Deployment ~months
Low utilization
Not portable

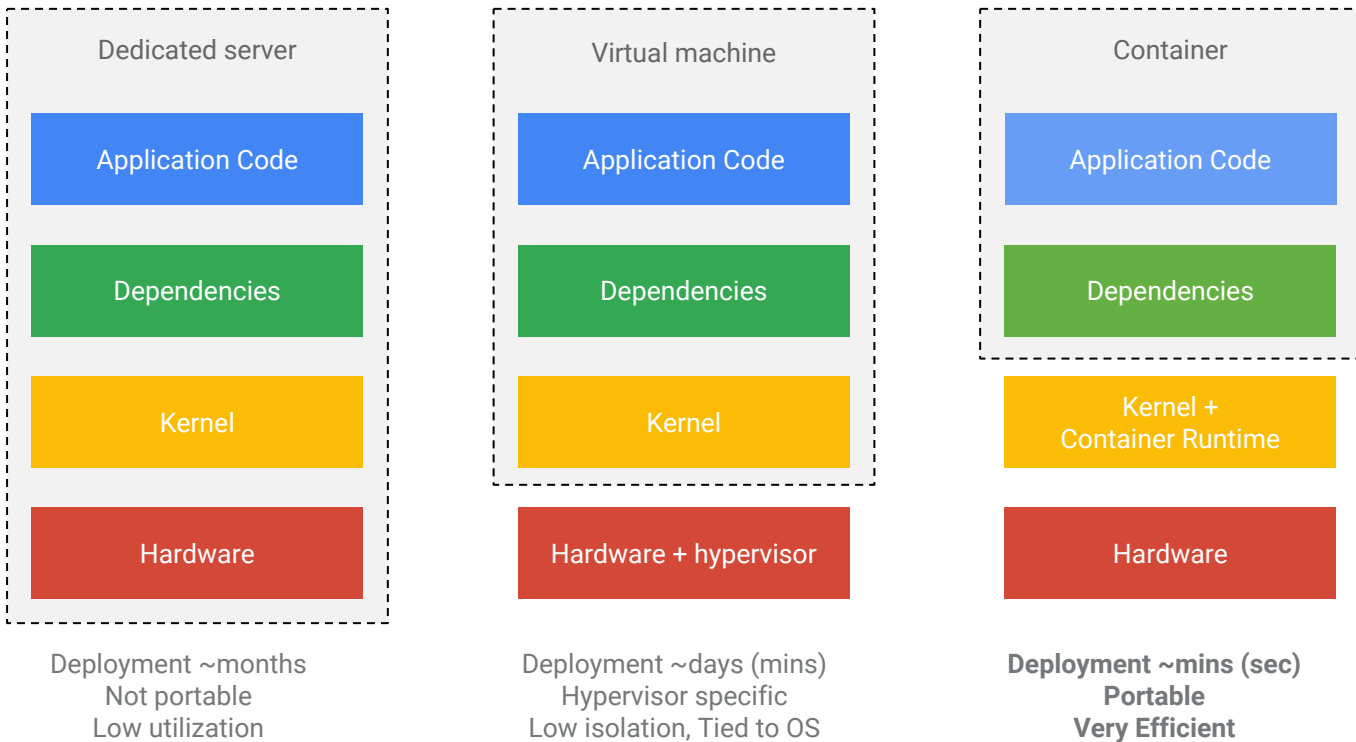


Deployment ~days (mins)
Hypervisor specific
Low isolation, Tied to OS

Many Virtual Machines

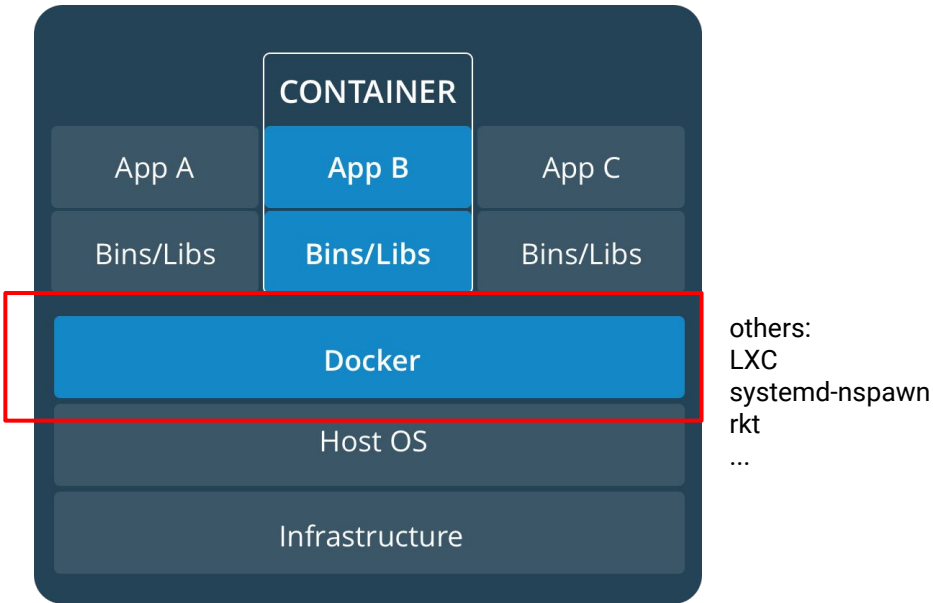


Containers



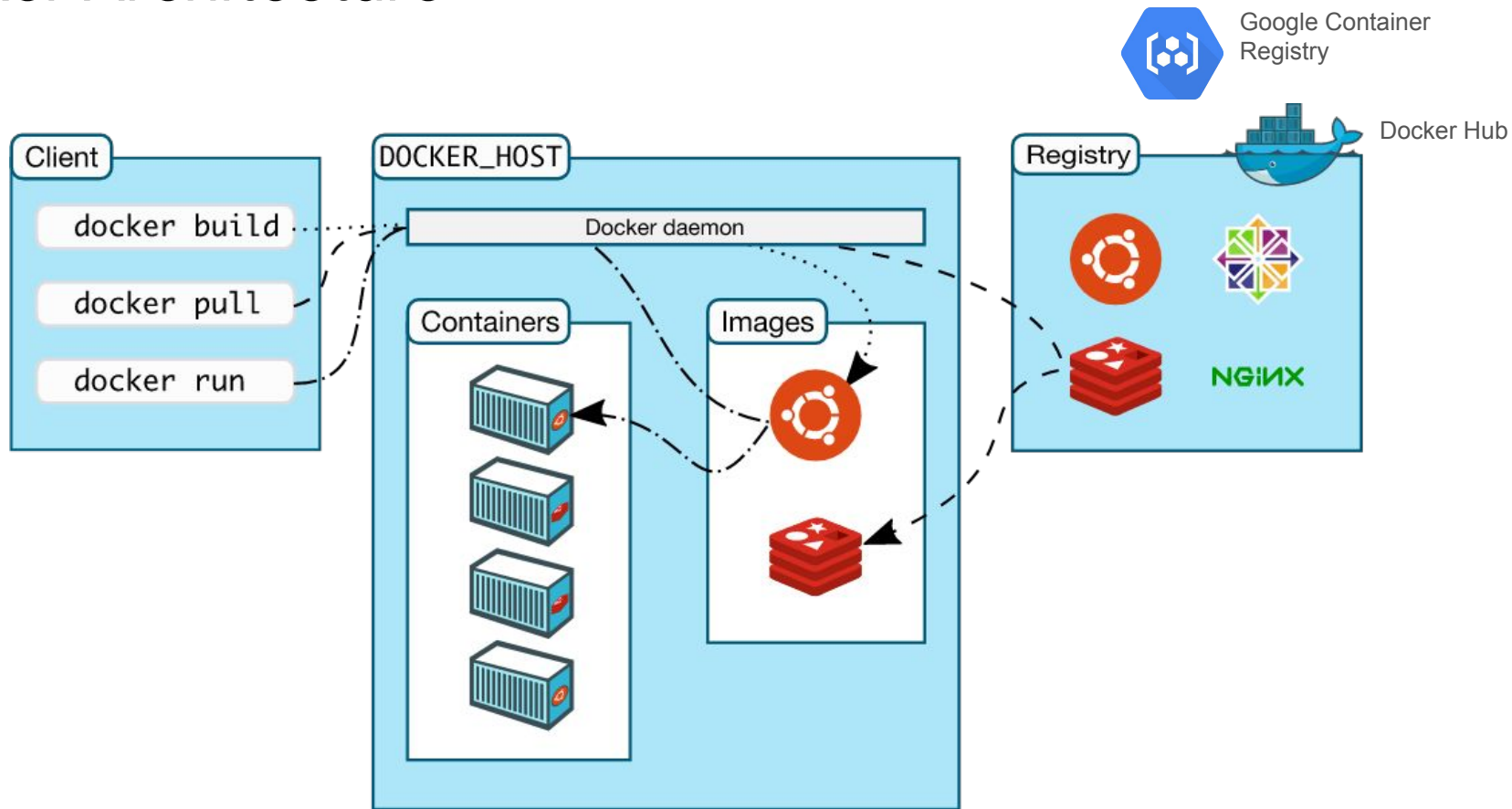
What is Docker?

- Docker is a software platform for building and running containers.
- The Docker Engine is the container runtime that builds, runs, and monitors containers.
- It uses underlying Linux kernel features such as cgroups, namespaces, and copy-on-write storage.
- Docker Engine is not the only container runtime. Other runtimes include LXC, systemd-nspawn, CoreOS rkt, and more.



*Images provided by Docker

Docker Architecture



Why containers?

"It ran fine on
MY machine"

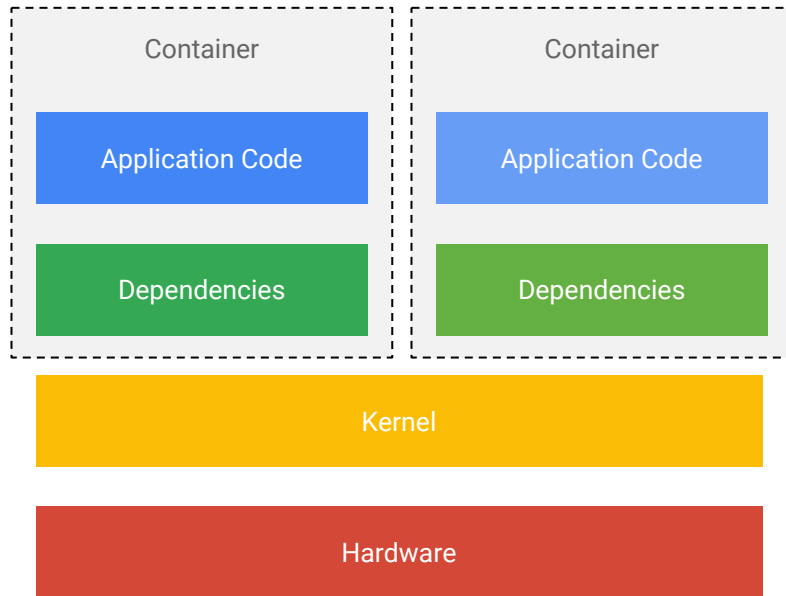
"We want to get the
best utilization of
our infrastructure"

"Keeping our
infrastructure perfectly
homogenous is
giving me nightmares"

"My developers aren't
as productive as they
should be. Deployments
are slowing us down"

Why containers?

- **Separation of code and compute**
 - Consistency across dev, test, and production
 - Consistency across bare-metal, VMs, and cloud
 - No more “it worked on my computer”
- **Packaged applications**
 - Agile application creation and deployment
 - Continuous Integration/Delivery
- **Conducive for Microservices**
 - Introspectable
 - Isolated/loosely coupled, conducive for distributed systems, and elastic



But, no such thing as a free lunch

"Where should I run my containers?"

"Distributed systems are hard."

"How do I get my containers to talk to one another?"

"How do we ensure our containers and the infrastructure they run on are healthy?"

"We don't want to be locked into one cloud provider"



LAB-1

Docker Fundamentals

The objective of this lab is to familiarize yourself with basic Docker commands. You will create, run, and debug containers on a VM. You will also learn to pull and push images to and from Google Cloud Registry.

Kubernetes

Google Kubernetes Engine

Agenda

Why Kubernetes?

Kubernetes Core Concepts

Kubernetes Cluster Architecture

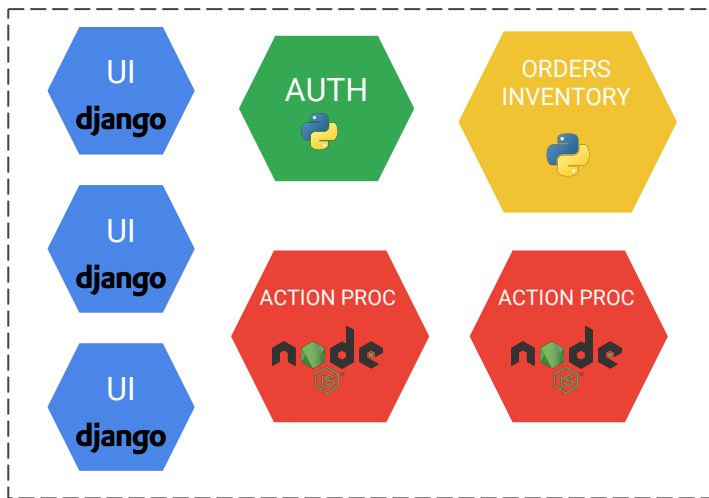
01

| Why Kubernetes?

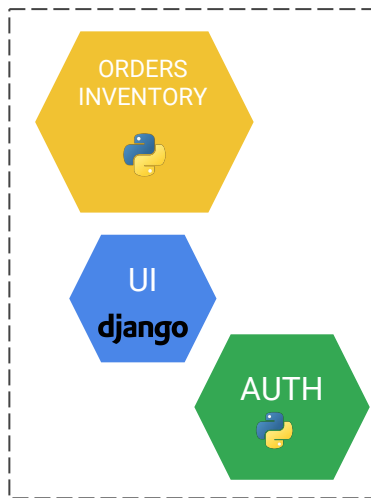


Can't call it a day with containers alone

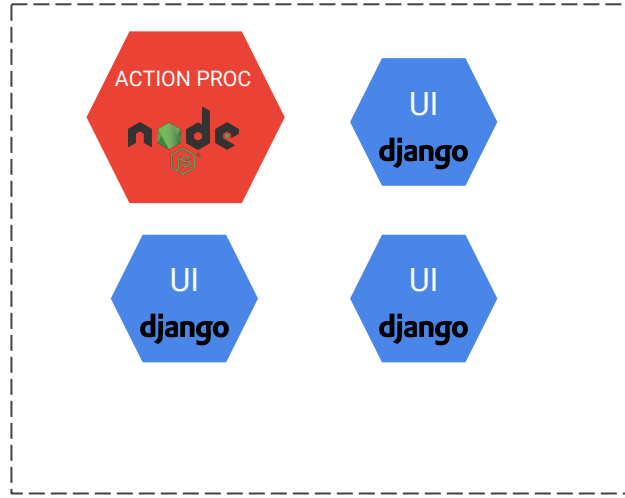
Node 1



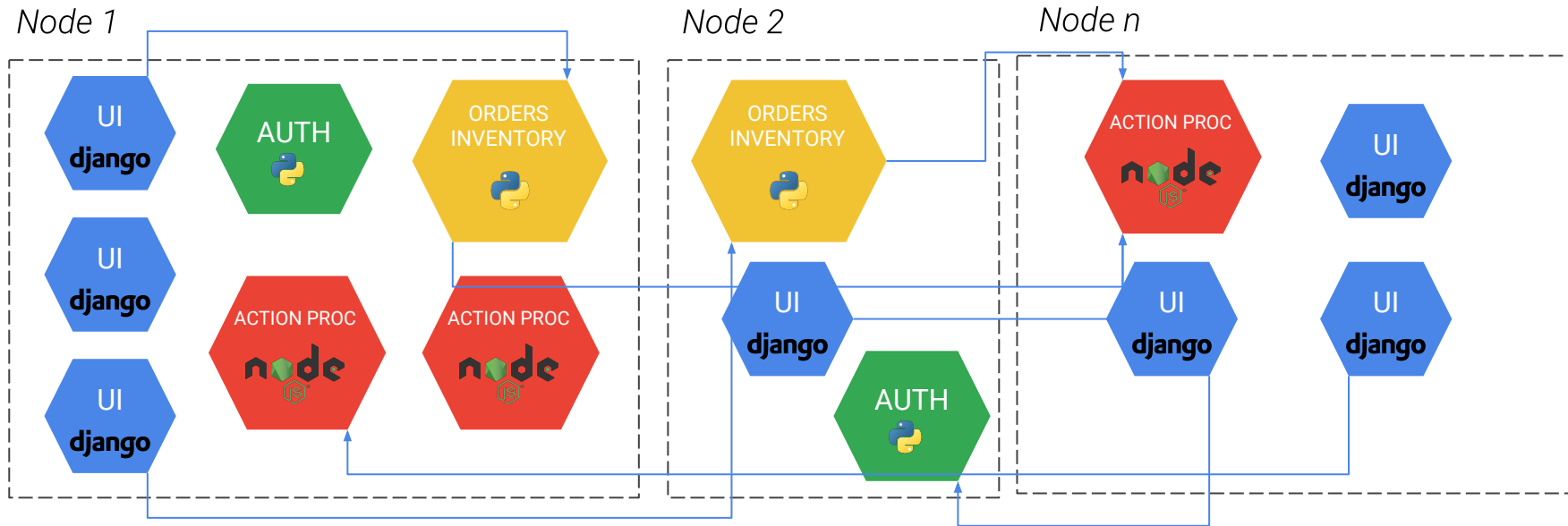
Node 2



Node n



Distributed systems are hard



Tenets of a good distributed system

Automated Scheduling

Health Checking

Various Deployment Methods

Load Balancing across Containers

Centralized Logging

Centralized Monitoring

Automated Binpacking

Responsive Autoscaling

Managing Secrets

Managing Application Configuration

Built-in Service Discovery

Rolling back Deployments

Responsive Autoscaling

Accommodating Different QoS

Stable Networking/VIPs

Container Resiliency

Role Based Access Control

Primitives for Isolation

Mounting Storage Volumes

Long-Running v. Batch Apps

Enter Kubernetes.

Automated Scheduling

Health Checking

Various Deployment Methods

Load Balancing across Containers

Centralized Logging

Centralized Monitoring

Automated Binpacking

Responsive Autoscaling

Managing Secrets

Managing Application Configuration



kubernetes

Built-in Service Discovery

Rolling back Deployments

Responsive Autoscaling

Accommodating Different QoS

Stable Networking/VIPs

Container Resiliency

Role Based Access Control

Namespaces for Isolation

Mounting Storage Volumes

Long-Running v. Batch Apps

What is Kubernetes?



Eric Tune

@erictune4

Following



Kubernetes is two things to me: abstraction over infrastructure and set of declarative APIs.

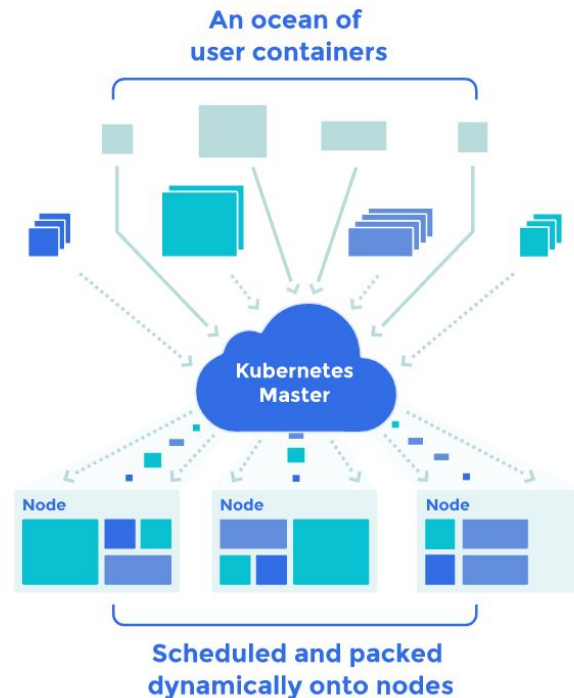
What is Kubernetes?

OSS Platform for Container Orchestration at Scale

- *Born from Google's learnings running billion user applications on Borg and Omega for 10+ years.*
- *Fun fact: Google still deploys ~2 billion containers/week on these systems.*

Declarative Management of Containerized Apps

- *Write a manifest declaring desired state of your app (cpu, mem, disk, load balancer, # of replicas & more) and Kubernetes will make sure it is running somewhere in a cluster of virtual machines.*



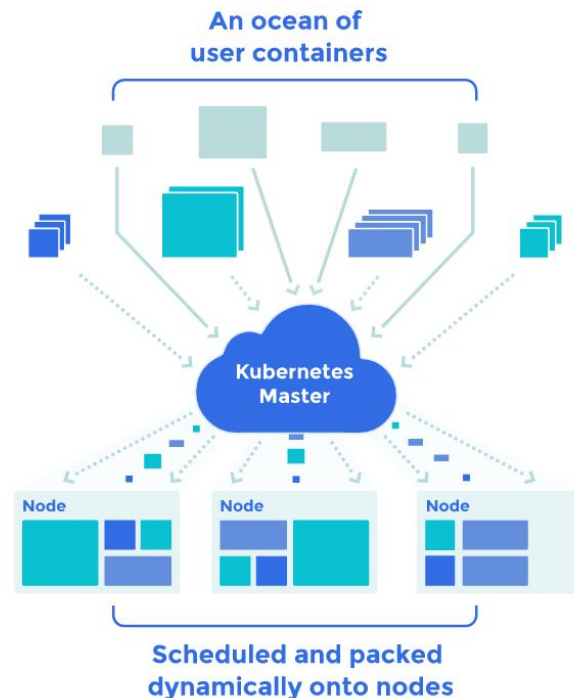
What is Kubernetes?

Decouples containers from hosts

- *Containers alone do not solve decoupling apps from awareness of the infrastructure.*
- *Kubernetes fully enables you to treat containers and apps like cattle and not pets.*

Coined by some as “Linux for Distributed Systems”

- *Linux abstracts away hardware differences.*
- *Kubernetes abstracts away the fact that you have hundreds if not thousands of nodes, giving you a consistent interface to run and operate apps on those nodes.*



What is Kubernetes?

Built in 2014 by 4 Googlers

- Major Releases every 3 months. Currently on v 1.9

Official CNCF Project

- Drew large enterprise members, including Amazon and Oracle

Written in Go and Organized by SIGs

- Special Interest Groups breaks down specific teams into specific functions for Kubernetes

Stylized as K8s, because...

K - ubernete (8 characters) - s

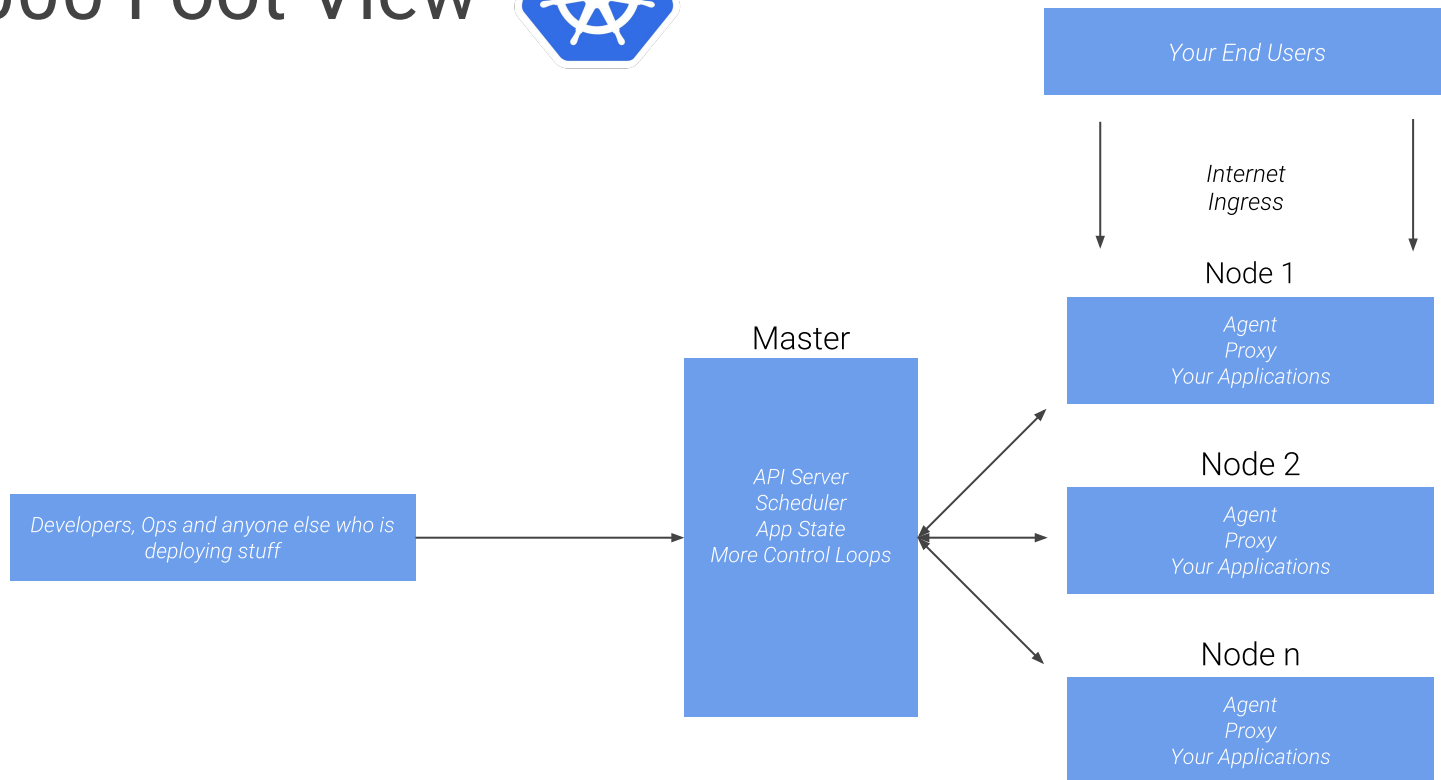


32 000+
pull requests
the latest year

60,000+
commits
the latest year

~23 PRs
merges/day
in the core repo

10,000 Foot View



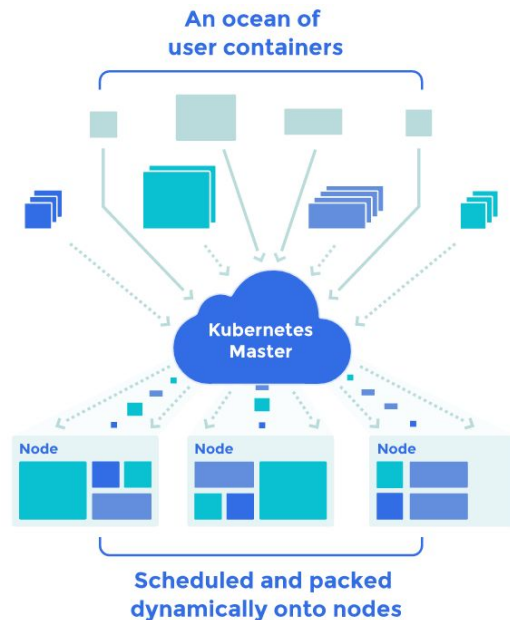
Kubernetes: Open Source Container Automation

- Open Source: Official CNCF Project, No. 1 Github Activity, Huge Community
- Container-Centric Infrastructure: No longer bound to hosts/VMs, provides relevant primitives
- Declarative Management: Define desired state, not workflow aka how to get from A -> B
- Application Resilience: Automates drivers to desired state without intervention
- Automatic Binpacking: Automates scheduling based on container resource requirements



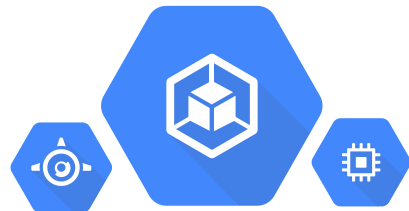
Kubernetes: Administration

- Provisioning Infrastructure: Create VM's and bootstrap core components for Kubernetes control plane
- Implement Networking: IP Ranges for Pods, Services, configuring an overlay
- Bootstrap Cluster Services: Provision kube-dns, logging daemons, monitoring
- Ensuring Node Resiliency: At a master & node level, ensuring uptime, configuring kernel upgrades, OS updates



Why use Google Kubernetes Engine?

- Deploying or maintaining a fleet of VMs has been a **challenge** and you've determined that **containers** are the solution.
- You've **containerized your workload (or on your way to)** and need a system on which to run and manage it.
- You **never want to touch (or would prefer to) a server or infrastructure.**
- You don't have **dependencies on kernel changes (or can do without them)** or on a specific (non-Linux) operating system.

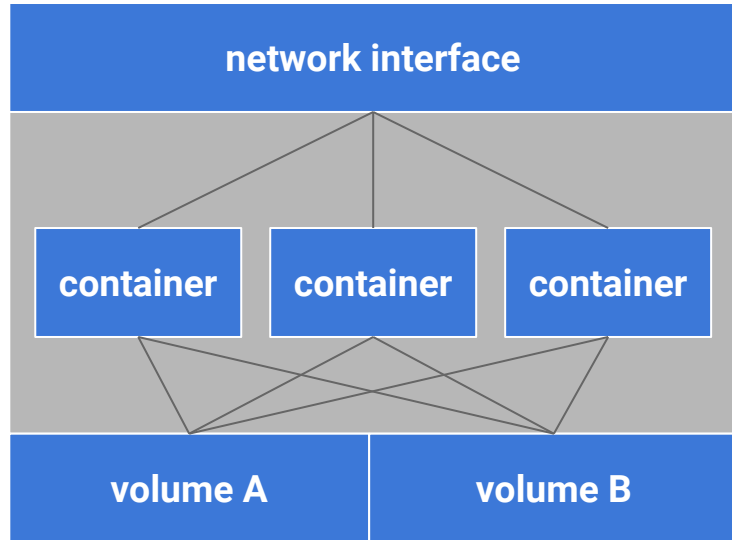


Kubernetes Engine

Cluster manager and
orchestration engine
built on Google's
container experience

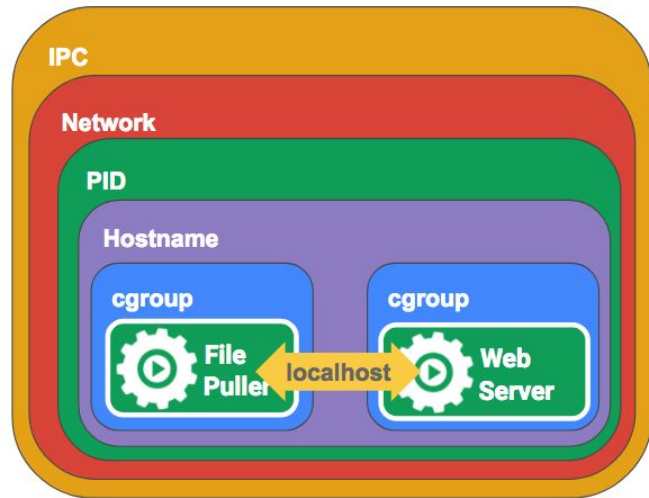
Core Concepts: The Pod

- Pod: The atomic unit of Kubernetes
- Comprised of one or more containers with shared networking & storage
- Containers in a pod share linux namespaces, but different control groups
- Kubernetes will nicely automate setting up namespace, cgroup
- Great for packaging containers together but...

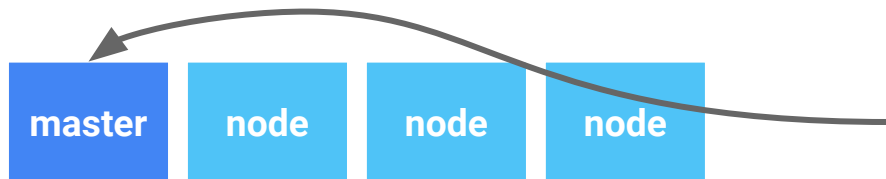


Core Concepts: The Pod

- Pod: The atomic unit of Kubernetes
- ...no native resiliency built into just a pod (we get that elsewhere)
- ...its IP is mortal (we will come back to that)
- ...can be used to group together different components (like a LAMP stack) but ideally is made for co-located helpers like a file loader or to watch something

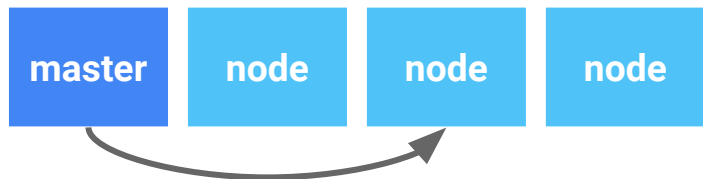


Core Concepts: The Pod (and manifest)



```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
    - name: mv-app
      image: my-app
    - name: nginx-ssl
      image: nginx
  ports:
    - containerPort: 80
    - containerPort: 443
```

Core Concepts: The Pod (and manifest)



```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
    - name: mv-app
      image: my-app
    - name: nginx-ssl
      image: nginx
  ports:
    - containerPort: 80
    - containerPort: 443
```

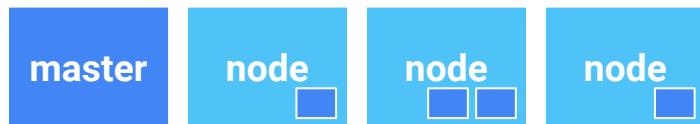
Core Concepts: The Pod (and manifest)



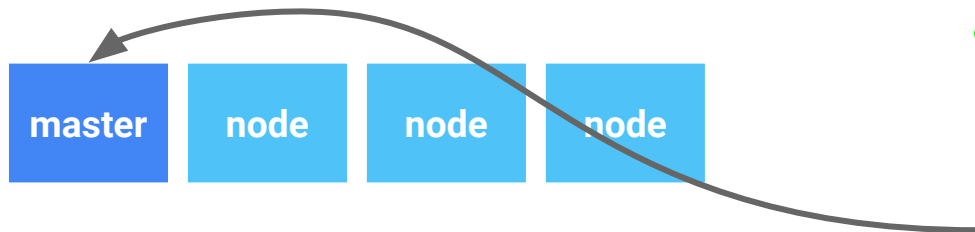
```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
    - name: mv-app
      image: my-app
    - name: nginx-ssl
      image: nginx
  ports:
    - containerPort: 80
    - containerPort: 443
```

Core Concepts: Deployments

- Deployment: An abstraction that allows you to define and update desired pod template and replicas
- If pods are mortal, abstractions like deployments give us resiliency
- One of many abstractions to control how pods are scheduled and deployed

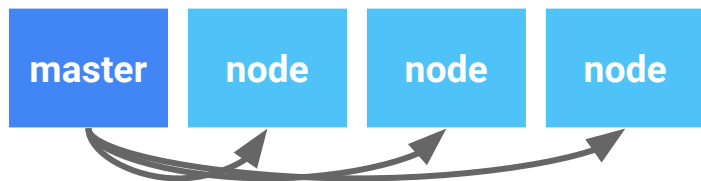


Core Concepts: Deployments



```
kind: Deployment
apiVersion: v1beta1
metadata:
  name: frontend
spec:
  replicas: 4
  selector:
    role: web
  template:
    metadata:
      name: web
      labels:
        role: web
    spec:
      containers:
        - name: my-app
          image: my-app
        - name: nginx-ssl
          image: nginx
      ports:
        - containerPort: 80
        - containerPort: 443
```

Core Concepts: Deployments



```
kind: Deployment
apiVersion: v1beta1
metadata:
  name: frontend
spec:
  replicas: 4
  selector:
    role: web
  template:
    metadata:
      name: web
      labels:
        role: web
    spec:
      containers:
        - name: my-app
          image: my-app
        - name: nginx-ssl
          image: nginx
      ports:
        - containerPort: 80
        - containerPort: 443
```

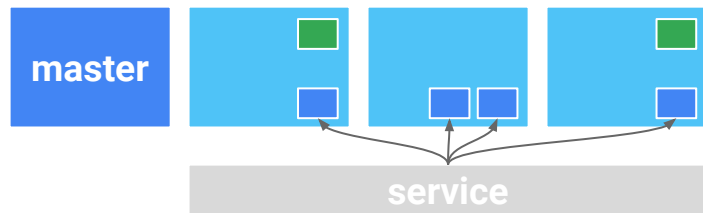
Core Concepts: Deployments



```
kind: Deployment
apiVersion: v1beta1
metadata:
  name: frontend
spec:
  replicas: 4
  selector:
    role: web
  template:
    metadata:
      name: web
      labels:
        role: web
    spec:
      containers:
        - name: my-app
          image: my-app
        - name: nginx-ssl
          image: nginx
      ports:
        - containerPort: 80
        - containerPort: 443
```

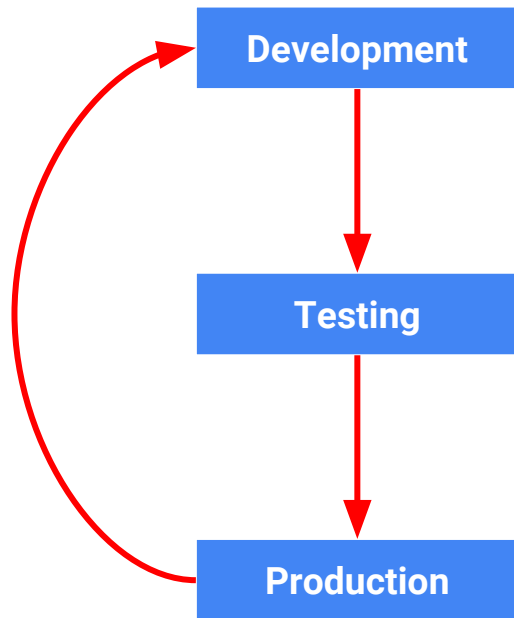
Core Concepts: Services

- Services: Stable endpoint for pods
- If pod IPs are mortal, services give us a stable way to access our pods
- Provides load balancing across multiple pods
- With services you can speak to pods via external IP, cluster internal IP or DNS
- Service will target multiple pods with the same kv-pair metadata, known as a label selector



Core Concepts: Namespaces

- Most organizations have a variety of different environments, such as production, staging, testing, development etc.
- They generally come either with strict access and security controls in terms of who can deploy what where, or else on the other end of the spectrum, they are wide open, with all users given free reign.

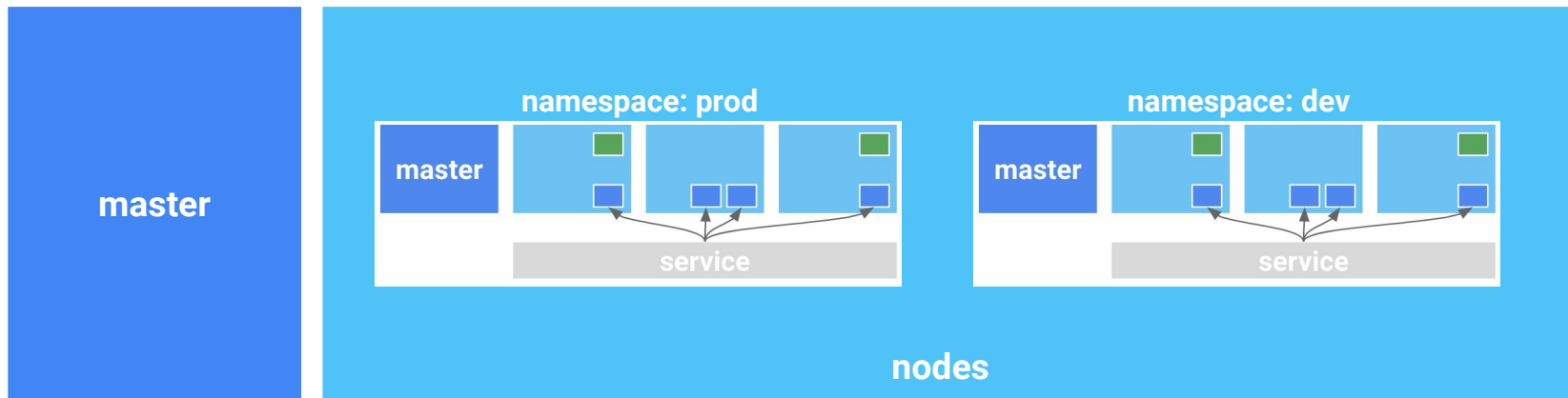


Kubernetes Components



Core Concepts: Namespaces

- Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces
- A Kubernetes namespace provides the scope for Pods, Services, and Deployments in the cluster.



Core Concepts: Namespaces

- Namespaces are intended for use in environments with many users spread across multiple teams, or projects.
- Users interacting with one namespace do not see the content in another namespace.

```
{  
  "kind": "Namespace",  
  "apiVersion": "v1",  
  "metadata": {  
    "name": "development",  
    "labels": {  
      "name": "development"  
    }  
  }  
}
```

```
{  
  "kind": "Namespace",  
  "apiVersion": "v1",  
  "metadata": {  
    "name": "production",  
    "labels": {  
      "name": "production"  
    }  
  }  
}
```

Create namespaces

Core Concepts: Namespaces

- Namespaces are intended for use in environments with many users spread across multiple teams, or projects.
- Users interacting with one namespace do not see the content in another namespace.

```
$ kubectl config set-context dev \  
  --namespace=development \  
  --cluster=main-cluster \  
  --user=kubs_user
```

```
$ kubectl config set-context prod \  
  --namespace=production \  
  --cluster=main-cluster \  
  --user=kubs_user
```

Switch **namespaces**, or contexts

Core Concepts: Context

- It defines a grouping of a cluster, a namespace, and a user
- Users interacting in one context do not see the content in another namespace

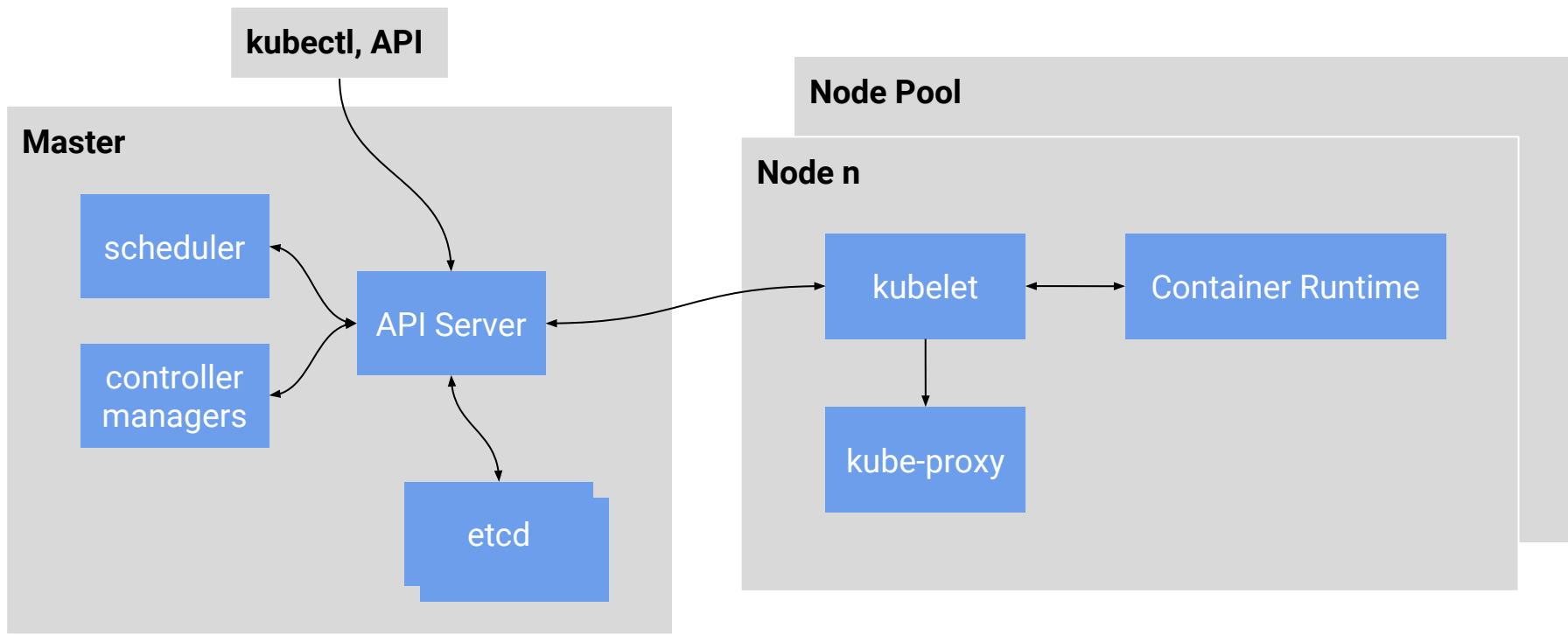
```
kubectl config --kubeconfig=config-demo \  
  set-context dev-frontend \  
  --cluster=development \  
  --namespace=frontend \  
  --user=developer
```

```
kubectl config --kubeconfig=config-demo \  
  set-context dev-storage \  
  --cluster=development \  
  --namespace=storage \  
  --user=developer
```

Add **context** details to your configuration file

Let's talk about K8 architecture...

Kubernetes Components



Clients: kubectl

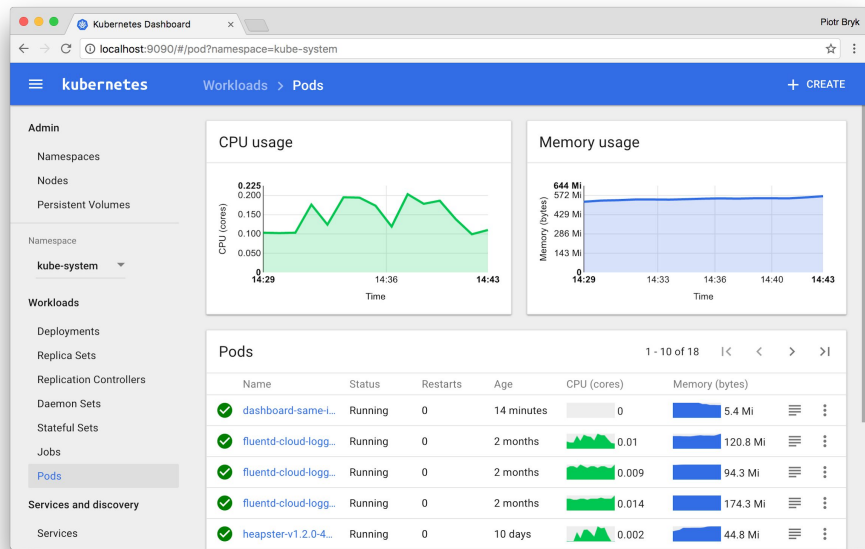
Kubectl is a CLI for running commands against Kubernetes Clusters and interacting with Kubernetes objects

kubectl	-n=kube-system	get	pod	yaml
		describe	deployment	json
		rm:delete	secret	owide
		logs	ingress	all
		exec	node	watch
		apply	svc	file
			ns	l
			cm	

Clients: Kubernetes Dashboard

Aims to make it simple to visualize cluster state

Shipped with cluster as a deployment and exposed via a Service





Hello Node Kubernetes

The goal of this hands-on lab is for you to turn code that you have developed into a replicated application running on Kubernetes, which is running on Kubernetes Engine. For this lab the code will be a simple Hello World node.js app.

Orchestrating the cloud with GKE

Agenda

Why Google Kubernetes Engine?

GKE Cluster Architecture

GKE Control Plane

GKE Cluster Features

GKE Integrations with Google Cloud Platform

Why **Google Kubernetes Engine** ?

Kubernetes: A Tale in Two Parts

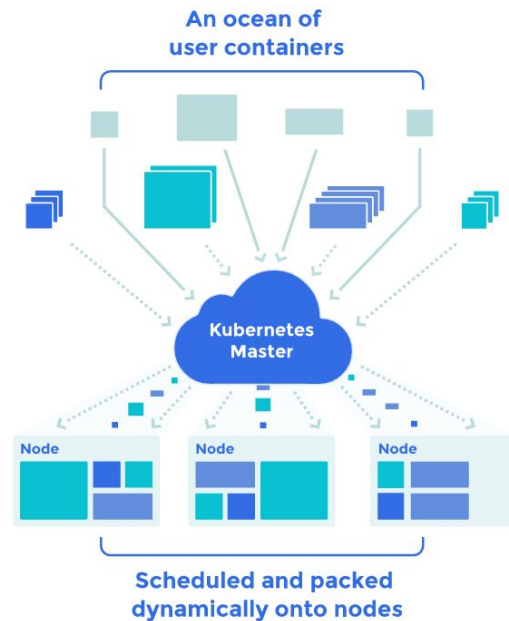
Part I: Cluster Administration

Provisioning Infrastructure: Create VM's and bootstrap core components for Kubernetes control plane

Implement Networking: IP Ranges for Pods, Services, configuring an overlay

Bootstrap Cluster Services: Provision kube-dns, logging daemons, monitoring

Ensuring Node Resiliency: At a master & node level, ensuring uptime, configuring kernel upgrades, OS updates

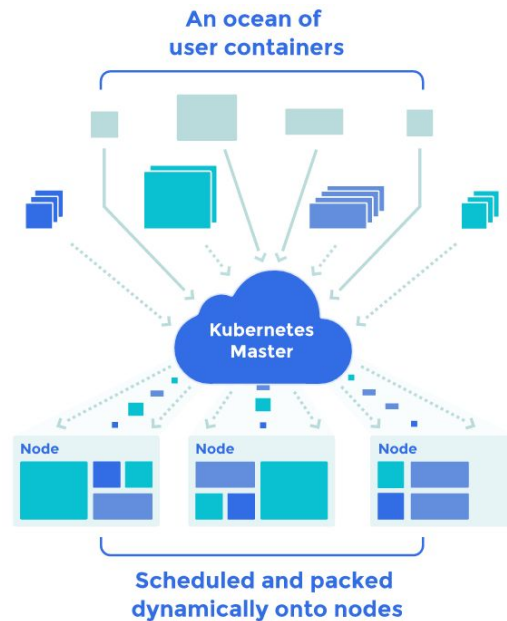


Kubernetes: A Tale in Two Parts

Part I: Cluster Administration

This helps us achieve ***abstraction over infrastructure***, at the cost of managing a complex system.

This can be challenging, but it's unavoidable. This is not necessarily where we want developers to use their time.



Kubernetes: A Tale in Two Parts

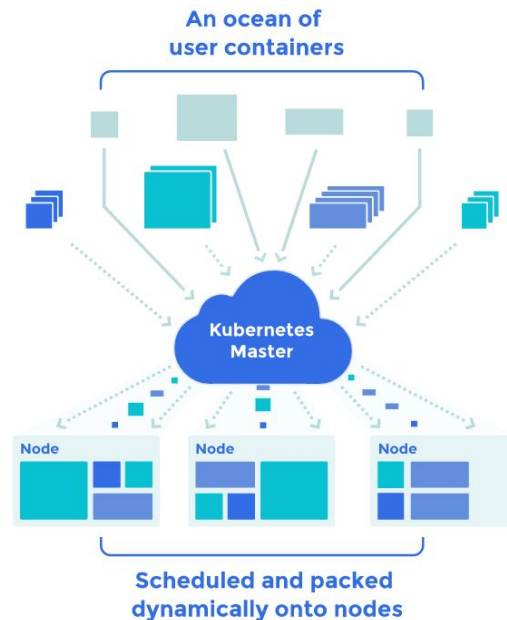
Part II: Actually *Using* the Cluster

Deploying Applications: Using Kubernetes constructs to deploy containerized applications

This is a distinctly different experience from cluster administration.

Developers should be focused on this paradigm of Kubernetes: using it as a ***set of declarative APIs***.

How can we enable this?



Enter Google Kubernetes Engine.

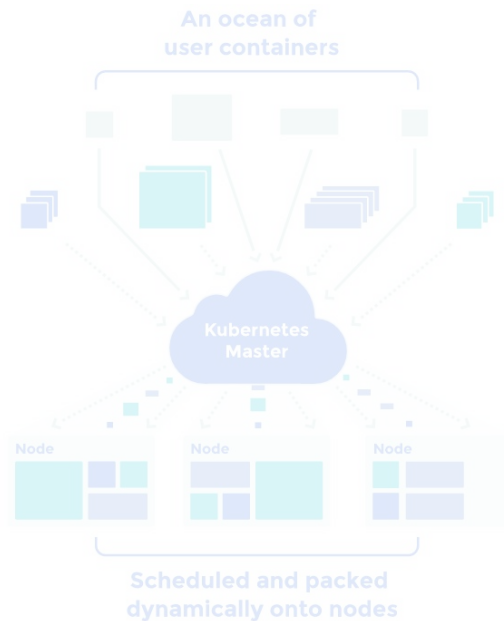
Part II: Actually *Using* the Cluster

Deploying Applications: Using Kubernetes constructs to deploy containerized applications

This is a distinctly different experience from administration.

Developers should be focused on this paradigm of Kubernetes: using it as a **set of declarative APIs**.

How can we enable this?



What is Google Kubernetes Engine?



Kelsey Hightower ✓

@kelseyhightower

Following



Google Container Engine is the cheat code for Kubernetes. Thanks to my fellow Googlers I get to focus on using Kubernetes; not managing it.

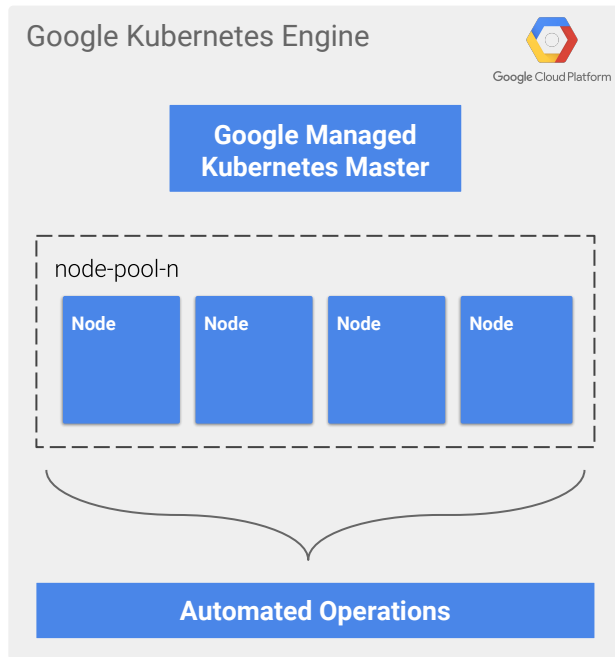
What is Google Kubernetes Engine?

Managed Kubernetes Cluster on Google Cloud Platform

Formerly known as Google Container Engine, **GKE is the easiest & fastest way to production Kubernetes**

gcloud container clusters create <cluster-name> **provisions a functional Kubernetes cluster in minutes**

Tracks OSS Kubernetes typically < 1 week after every minor release



02

| GKE Cluster Architecture



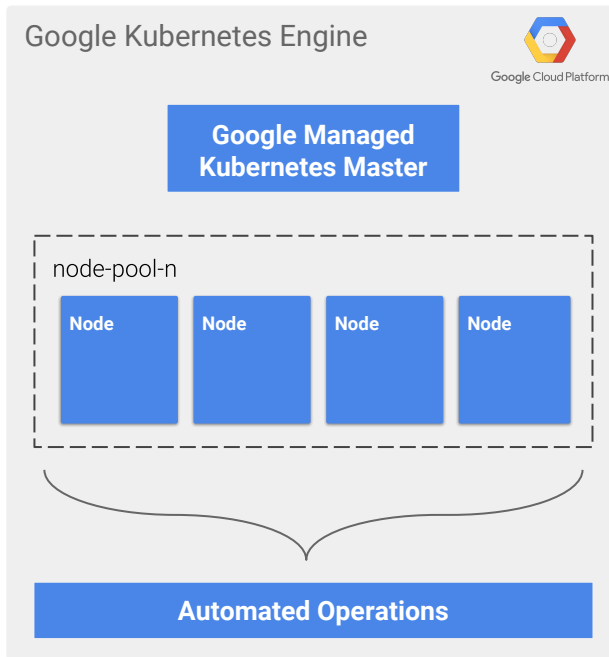
GKE Architecture

Fully-managed Kubernetes control plane backed by SLA with no-op for master maintenance

Automated operations tasks built-in, like **node upgrades/node repair**

Non-overlay **networking native to GCP**

Opinionated Kubernetes configuration



GKE Architecture: Master Node

Kubernetes Master backed by 99.5% SLO

Single VM running all control plane components in a **Google-managed project**

Upgrades are automatic, **can choose to opt-in to upgrade earlier**

Master scaled automatically to accommodate cluster size

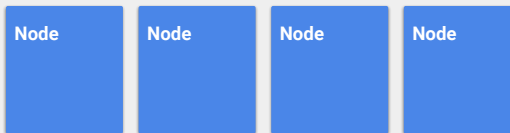
Master currently scoped to a single zone, **high-availability clusters are in Beta** to replicate the master across three zones in a single region, increasing the SLO to **99.99%**.

Google Kubernetes Engine



Kubernetes Master
us-central1-a

Node-pool-n, us-central1-a



GKE Architecture: Cluster Nodes

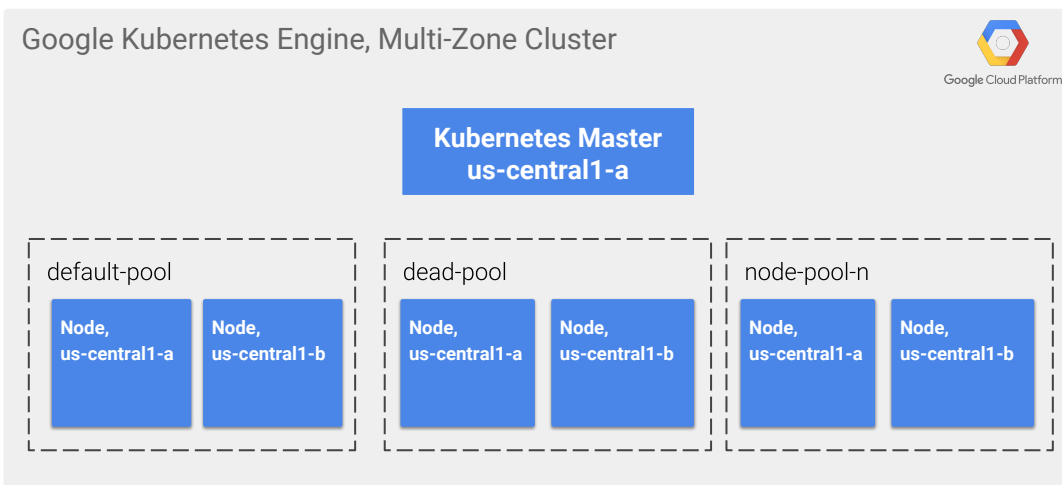
Managed by **node pools**, a group of nodes with **like configuration**

Can configure **one or multiple node pools** per cluster

Can use node pools for regular VM's or **preemptible VM's**

Even with a single zone master, can configure **multi-zone** node pools

Controlled by **Managed Instance Groups**, a construct in Google Compute Engine



GKE Architecture

What ships with a Kubernetes Engine Cluster?

Master Node (API Server, etcd Scheduler, Controller-Managers)

Nodes running Container Optimized-OS or Ubuntu

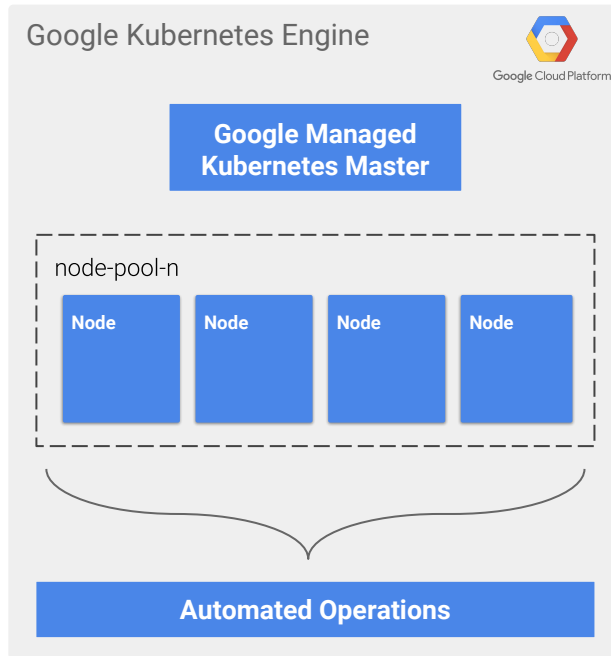
Kube-dns deployment

Kube-dns autoscaler deployment

Event-exporter deployment logging to Stackdriver Logging

fluentd daemon set logging to Stackdriver Logging

Heapster deployment writing to Stackdriver Monitoring



03

| GKE Control Plane



Managing the Kubernetes Master

From deployment to running, challenges can include...

- Managing components on master node
- Encrypting and securing etcd
- Rolling out security patches
- Ensuring uptime of the control plane
- Bootstrapping TLS
- Scaling master with cluster
- Configuring HA



Managing the Kubernetes Master

Kubernetes Engine enables you to use Google as your SRE for...

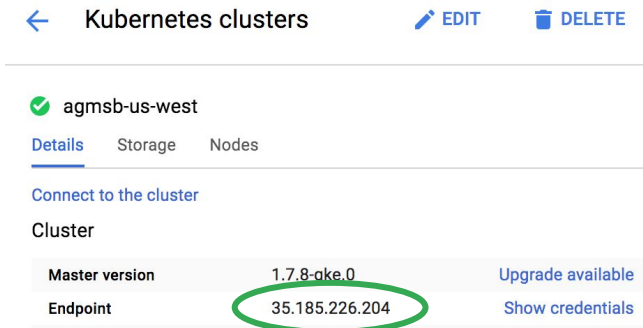
- Managing components on master node
- Encrypting and securing etcd
- Rolling out security patches
- Ensuring uptime of the control plane
- Bootstrapping TLS
- Scaling master with cluster
- Configuring HA



Managing the Kubernetes Master

With a managed Master comes a few considerations:

- **The Master has a public IP** that cannot be disabled.
- Kubernetes Engine provides **IP Rotation** to **obfuscate the master location**, rotating SSL certs & updating CA while provisioning a new public IP.
- Kubernetes Engine also provides **Master Authorized Networks**, which allow you to **whitelist a range of IPs that can access your Kubernetes master**.



The screenshot shows the Google Cloud console interface for managing Kubernetes clusters. At the top, there's a navigation bar with a back arrow, the text 'Kubernetes clusters', and 'EDIT' and 'DELETE' buttons. Below this, the cluster 'agmsb-us-west' is selected, with a green checkmark icon. Underneath, there are tabs for 'Details', 'Storage', and 'Nodes', with 'Details' being the active tab. A link 'Connect to the cluster' is visible. Below that, a table lists cluster details:

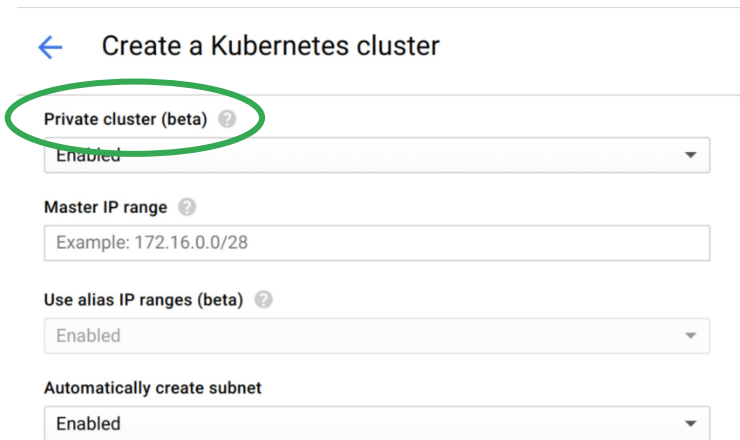
Master version	1.7.8-gke.0	Upgrade available
Endpoint	35.185.226.204	Show credentials

The 'Endpoint' value '35.185.226.204' is circled in green in the original image.

Managing the Kubernetes Master

With a managed Master comes a few considerations:

- In a default cluster, **the Master communicates with nodes via public IPs.**
- In a private cluster, **the Master communicates with nodes via private IPs** using VPC peering.
- In a private cluster, the Master **still has a public IP** but a **firewall disables external access** except for IP ranges from Master Authorized Networks and Google.



← Create a Kubernetes cluster

Private cluster (beta) ?

Enabled

Master IP range ?

Example: 172.16.0.0/28

Use alias IP ranges (beta) ?

Enabled

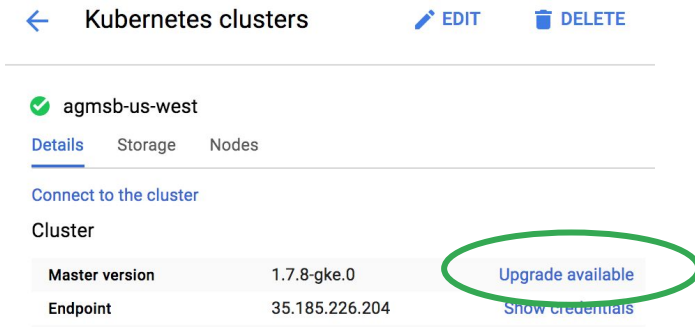
Automatically create subnet

Enabled

Managing the Kubernetes Master

With a managed Master comes a few considerations:

- **The Master is automatically upgraded by Google**
- Kubernetes Engine allows you to **opt-in and trigger the upgrade** once it is available in the UI.
- Kubernetes Engine allows you to define **maintenance windows** as to when operations like Master upgrade should occur.



← Kubernetes clusters [EDIT](#) [DELETE](#)

✓ agmsb-us-west

[Details](#) [Storage](#) [Nodes](#)

[Connect to the cluster](#)

Cluster

Master version	1.7.8-gke.0	Upgrade available
Endpoint	35.185.226.204	Show credentials

04

| GKE Cluster Features



GKE Cluster Features

Ease of Management / Reliability / Performance / Security / User Experience

Node Auto Upgrades: Automatically upgrade K8s components on nodes

Ensures node components are upgraded while also patching node image

Requires opt-in to have cluster with node auto-upgrade, subject to maintenance windows

Nodes drained, marked as unschedulable, and replaced one at a time

Manual node upgrade is also easier, via one-click process in UI

Best Practice: Ensure pods are managed by a controller. Use pod disruption budget to avoid service downtime. Don't let your nodes track behind the master more than two minor releases! Use persistent volumes if you need state as boot disks are deleted.

GKE Cluster Features

Ease of Management / Reliability / Performance / Security / User Experience

Node Auto Repair: Easily keep nodes in your cluster healthy and in a running state

Repairs node if it reports NotReady status or reports no status whatsoever

Repairs node if its boot disk is out of space for an extended period of time

Requires opt-in to use auto-repair

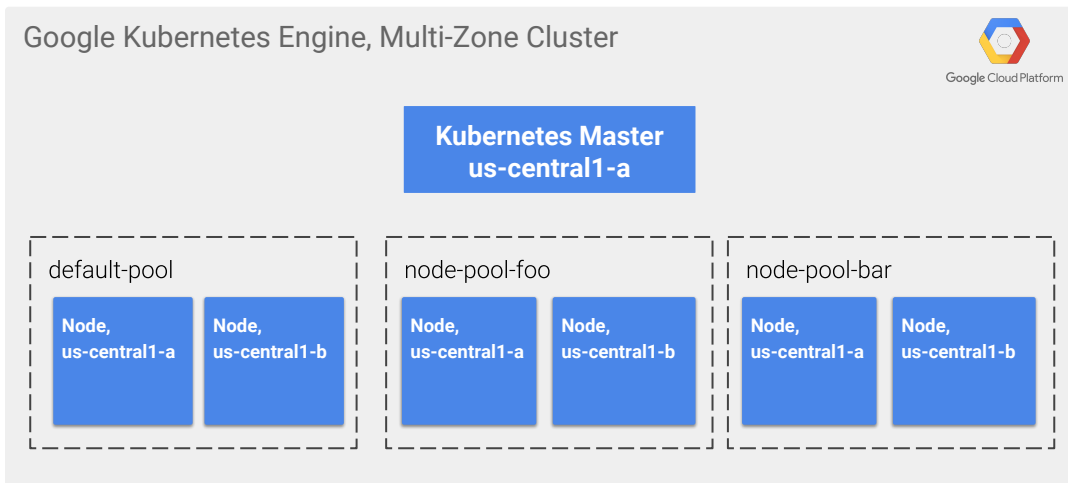
Repair process: Drains node and then recreates VM

Tips: If your nodes terminate, recreation will be handled whether or not you have node auto-repair enabled as it is controlled by the Node Pool/Managed Instance Group.

GKE Cluster Features

Ease of Management / **Reliability** / Performance / Security / User Experience

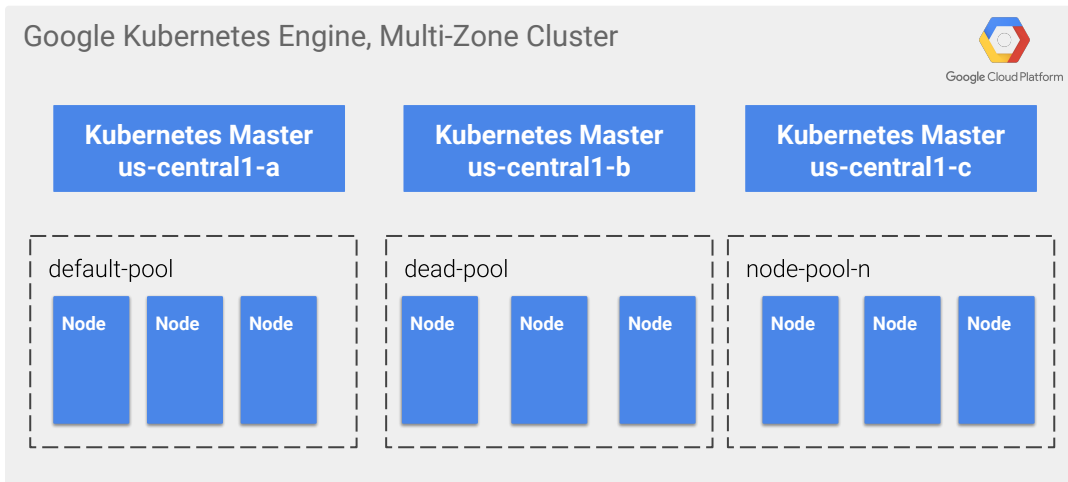
Multi-Zone Clusters: Enables higher service level by deploying nodes across multiple zones



GKE Cluster Features

Ease of Management / **Reliability** / Performance / Security / User Experience

Regional Clusters: Enables higher service level by deploying masters across multiple zones



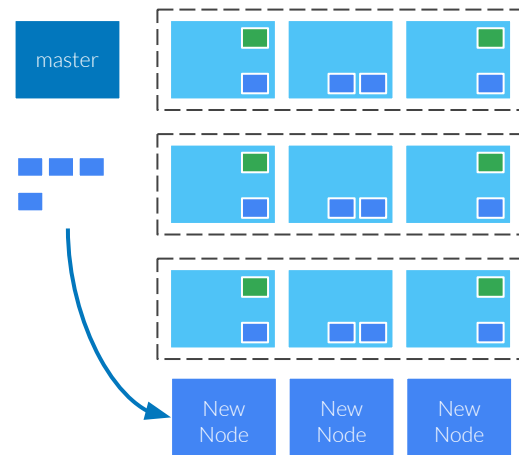
GKE Cluster Features

Ease of Management / Reliability / **Performance** / Security / User Experience

Cluster Autoscaler: Responsively scale out nodes in your Kubernetes Engine cluster

Cluster Autoscaler will add nodes when pods are failing to be scheduled due to insufficient resources (IE CPU/Memory).

Kubernetes uses requests to schedule pods, therefore scaling will occur based on pod resource requests not pod resource utilization



GKE Cluster Features

Ease of Management / Reliability / Performance / **Security** / User Experience

Encryption at Rest

All GKE components are encrypted at rest. This includes etcd, where secrets are stored.

TLS bootstrapped for you for node-master communication.

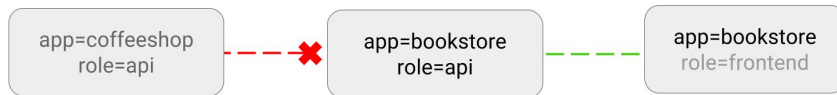
TLS enabled for Regional Clusters for master-master communication.

GKE Cluster Features

Ease of Management / Reliability / Performance / **Security** / User Experience

Network Policy: Securing Pod to Pod Communication

Integrates with Calico to provide default deny communication in a namespace, and allows you to use labels to whitelist pod to pod communication.



GKE Cluster Features

Ease of Management / Reliability / Performance / **Security** / User Experience

Cloud IAM

Provides roles at a project level across all Kubernetes Engine clusters in that project

Separate from Kubernetes native RBAC

Looking to achieve parity in the future

Role	Description
<code>roles/container.admin</code>	Full management of Container Clusters and their Kubernetes API objects.
<code>roles/container.clusterAdmin</code>	Management of Container Clusters.
<code>roles/container.developer</code>	Full access to Kubernetes API objects inside Container Clusters.

GKE Cluster Features

Ease of Management / Reliability / Performance / Security / **User Experience**

Native GCP Console for Kubernetes Engine Workloads

Dashboard for users to view Kubernetes objects from the GCP Console

This includes pods, load balancers, volumes and configuration.

Workloads BETA [REFRESH](#)

Workloads are deployable units of computing that can be created and managed in a cluster.

Filter workloads

Name	Status	Type	Pods	Namespace	Cluster
event-exporter-v0.1.7	OK	Deployment	1/1	kube-system	agmb-us-west
event-exporter-v0.1.7	OK	Deployment	1/1	kube-system	agmb-asia-east
event-exporter-v0.1.7	OK	Deployment	1/1	kube-system	agmb-eu-west
fluentd-gcp-v2.0.9	OK	Daemon Set	3/3	kube-system	agmb-us-west
fluentd-gcp-v2.0.9	OK	Daemon Set	3/3	kube-system	agmb-asia-east
fluentd-gcp-v2.0.9	OK	Daemon Set	3/3	kube-system	agmb-eu-west
heapster-v1.4.3	OK	Deployment	1/1	kube-system	agmb-us-west
heapster-v1.4.3	OK	Deployment	1/1	kube-system	agmb-asia-east
heapster-v1.4.3	OK	Deployment	1/1	kube-system	agmb-eu-west
kube-dns	OK	Deployment	2/2	kube-system	agmb-us-west
kube-dns	OK	Deployment	2/2	kube-system	agmb-asia-east
kube-dns	OK	Deployment	2/2	kube-system	agmb-eu-west
kube-dns-autoscaler	OK	Deployment	1/1	kube-system	agmb-us-west
kube-dns-autoscaler	OK	Deployment	1/1	kube-system	agmb-asia-east
kube-dns-autoscaler	OK	Deployment	1/1	kube-system	agmb-eu-west

05

| GKE Integrations with GCP



GKE Integration with GCP

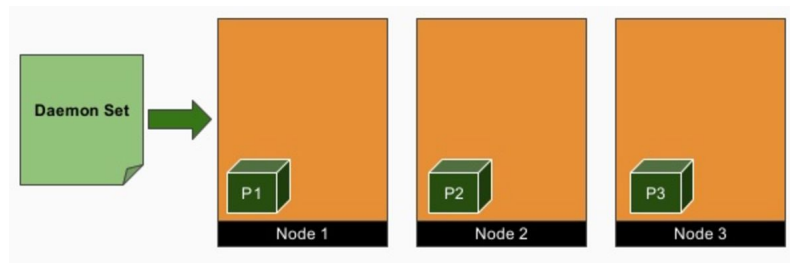
Logging & Monitoring / Load Balancing / Storage / Infrastructure

Automated deployment of logging and monitoring with Stackdriver

fluentd daemonSet writes container logs to Stackdriver (can use a configMap to write to other backends)

Heapster writes resource utilization collected from cAdvisor to Stackdriver

Event-exporter writes cluster events to Stackdriver



GKE Integration with GCP

Logging & Monitoring / **Load Balancing** / Storage / Infrastructure

Integrated with GCP's L4/L7 Fully-Managed Load Balancers

Services will provision a L4 Network Load Balancer

With an annotation in the service .yaml, you can provision an Internal Load Balancer

GLBC controller enables you to provision an L7 HTTP/S Load Balancer fulfilling Ingress

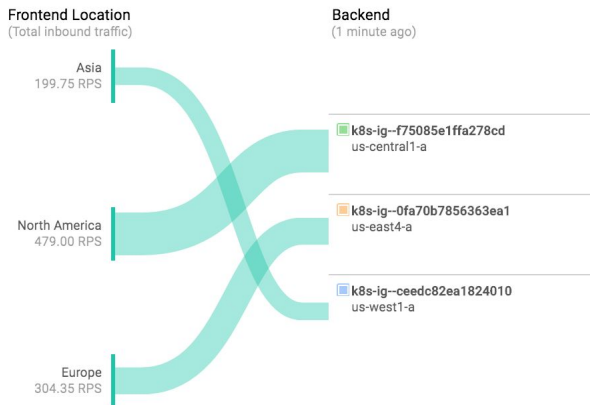
GKE Integration with GCP

Logging & Monitoring / **Load Balancing** / Storage / Infrastructure

Kubemci: Configuring Multi-Cluster Ingress

In Beta, you can use Kubemci to provision a single HTTP/S Load Balancer to route between multiple clusters based on geo-region

A popular use case for Federation, GKE is looking to bring Federation use cases native to Kubernetes Engine. Kubemci is just the start.



GKE Integration with GCP

Logging & Monitoring / Load Balancing / **Storage** / Infrastructure

Integrate with GCP Persistent Disk, SSD and LocalSSD

Can provision volumes by either creating disk and persistent volume/persistent volume claim, or by using a dynamic volume.

Can also utilize LocalSSD at cluster creation time, physically mounting SSD to hosts in your Kubernetes Engine Cluster

GKE Integration with GCP

Logging & Monitoring / Load Balancing / Storage / **Infrastructure**

Compute Engine: Quick Provisioning and Various Machine Types

GCE VM's typically have sub-minute boot times, this leads to GKE clusters provisioning in an order of a few minutes

Can also utilize Preemptible VM's to optimize cost if pods can tolerate eviction and rescheduling



LAB-3

Orchestrating the Cloud with Kubernetes

- Provision a complete Kubernetes cluster using Kubernetes Engine.
- Deploy and manage Docker containers using kubectl.
- Break an application into microservices using Kubernetes' Deployments and Services.

Contents

1. **Containers**
2. **Kubernetes**
3. **Google Kubernetes Engine**
4. Kubeflow - Overview
5. Kubeflow - Components
6. Kubeflow - Architecture
7. ksonnet
8. Kubeflow - Tutorial

Appendix- Tensor2Tensor

Kubeflow

Overview

Objectives



Composability



Portability

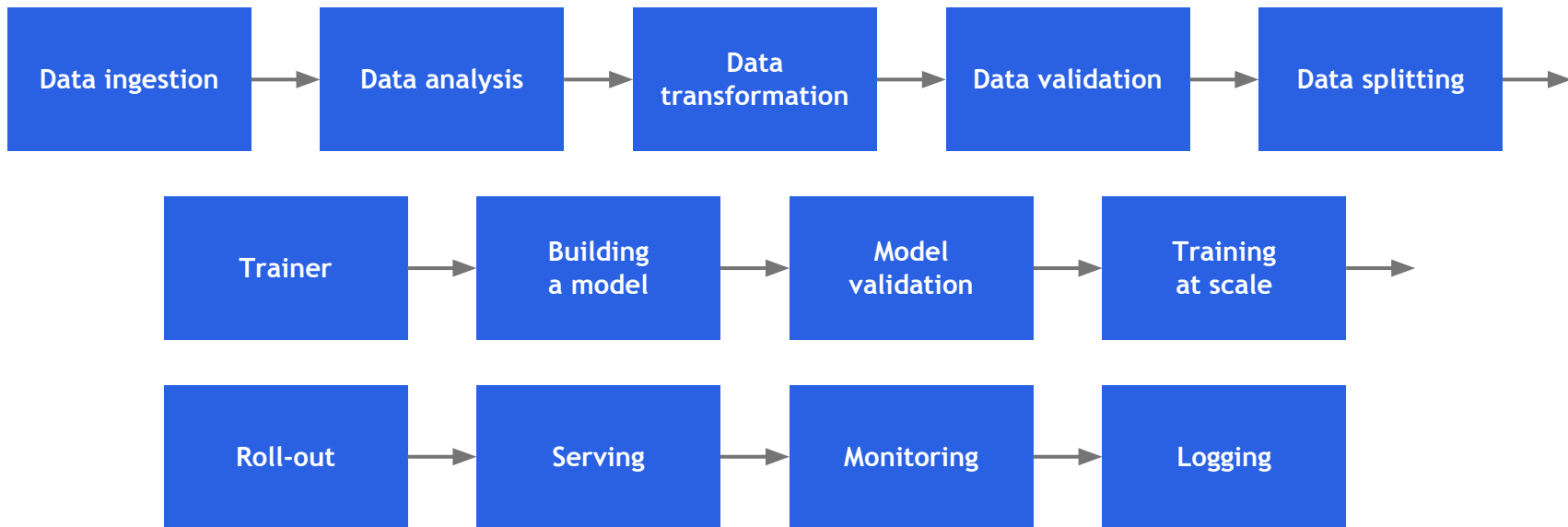


Scalability

Composability

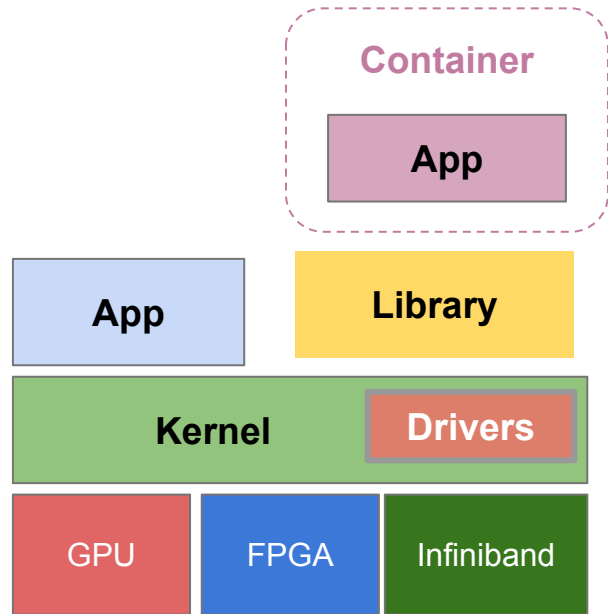
Building a model

Composability



Portability

- As a data scientist, you want to use the right HW for the job
- Every variation causes lots of pain
 - GPUs/FPGAs, ASICs, NICs
 - Kernel drivers, libraries, performance
- Even within an ML frameworks dependencies cause chaos
 - Tensorflow innovation (XLA)
 - ML portability is a challenge

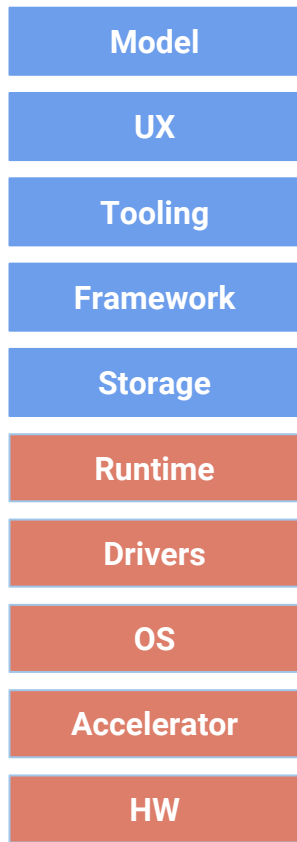
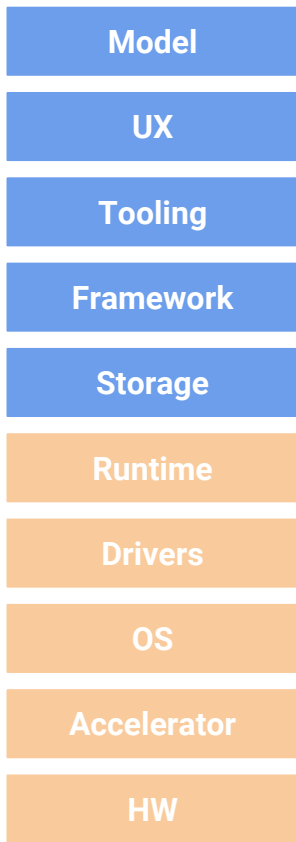
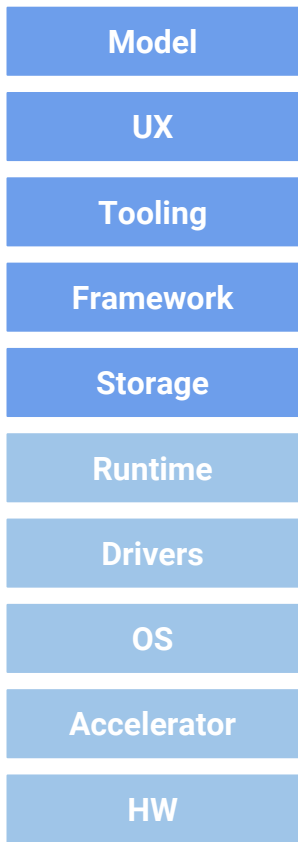
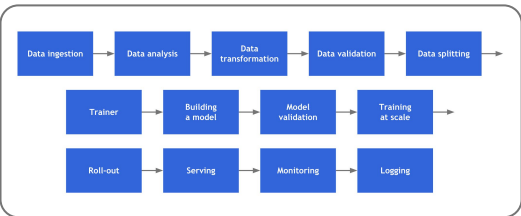


Portability

Laptop

Training Rig

Cloud

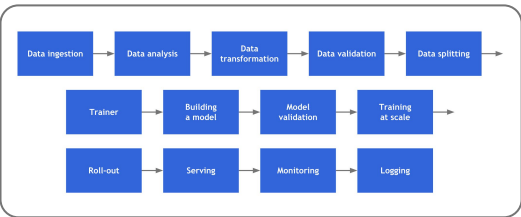
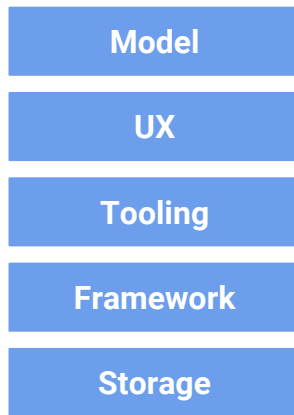


Portability

Laptop

Training Rig

Cloud

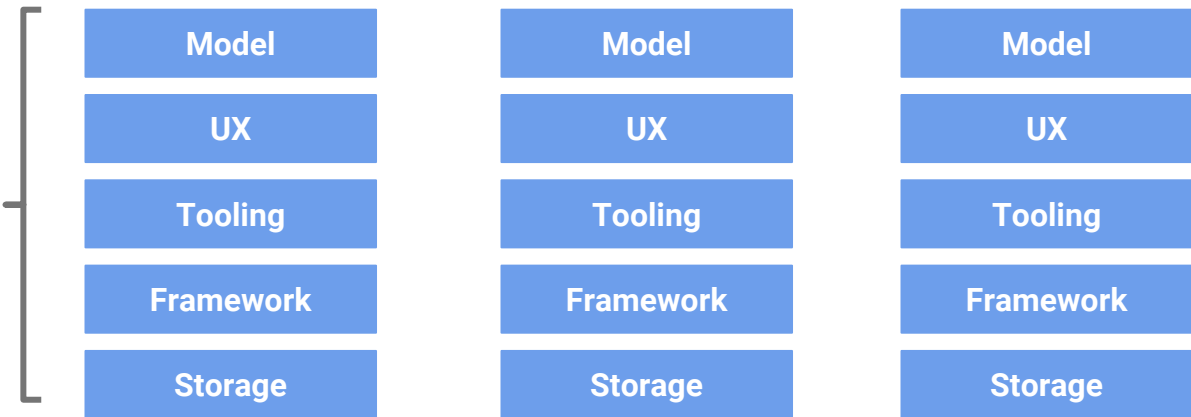


Portability

Laptop

Training Rig

Cloud

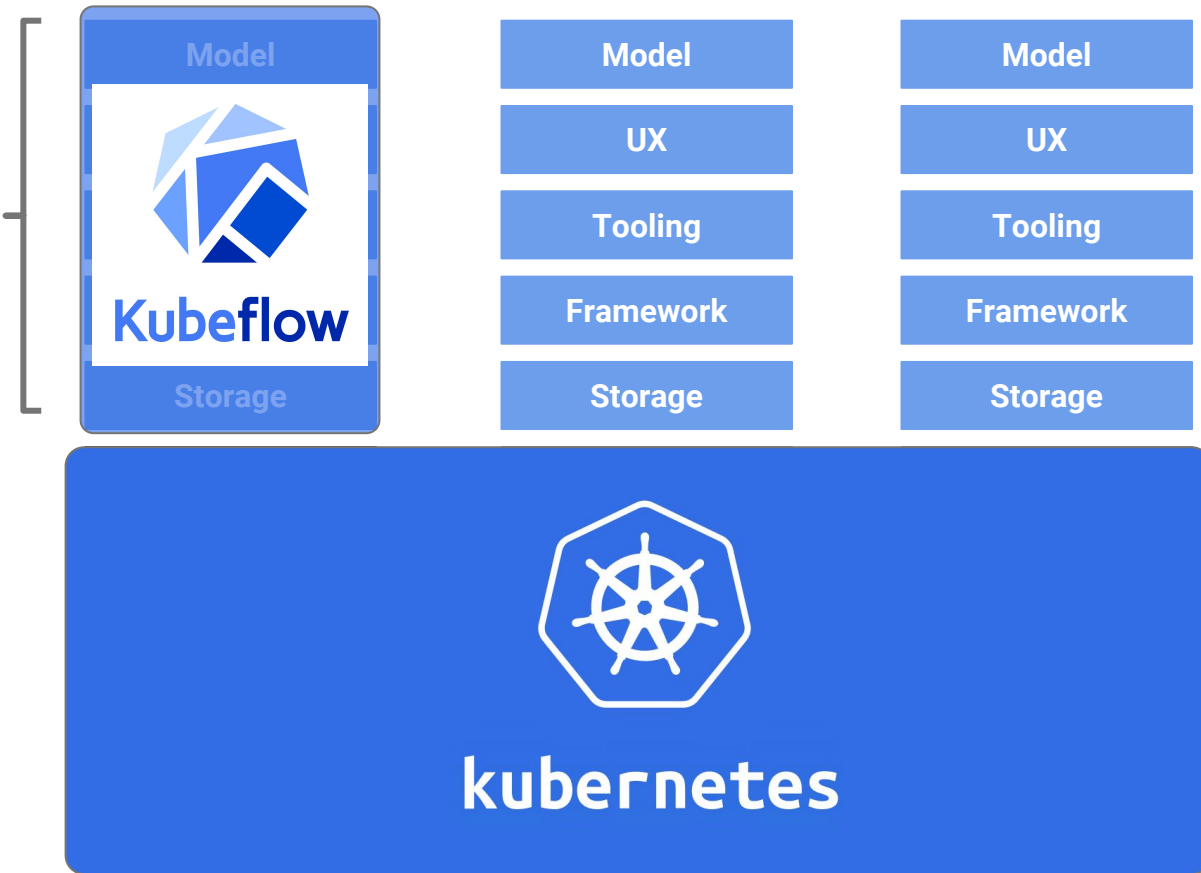


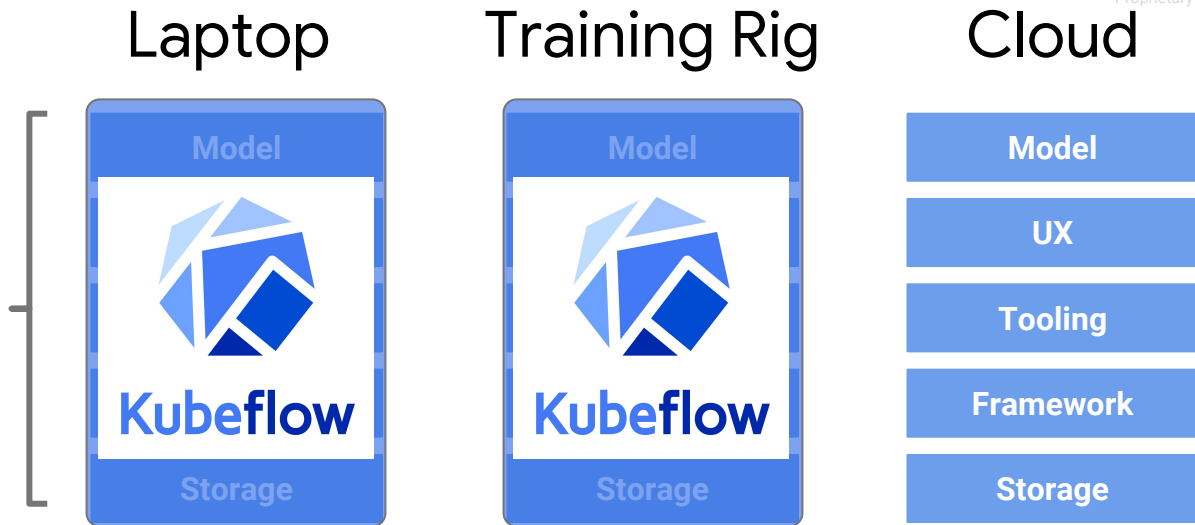
Portability

Laptop

Training Rig

Cloud



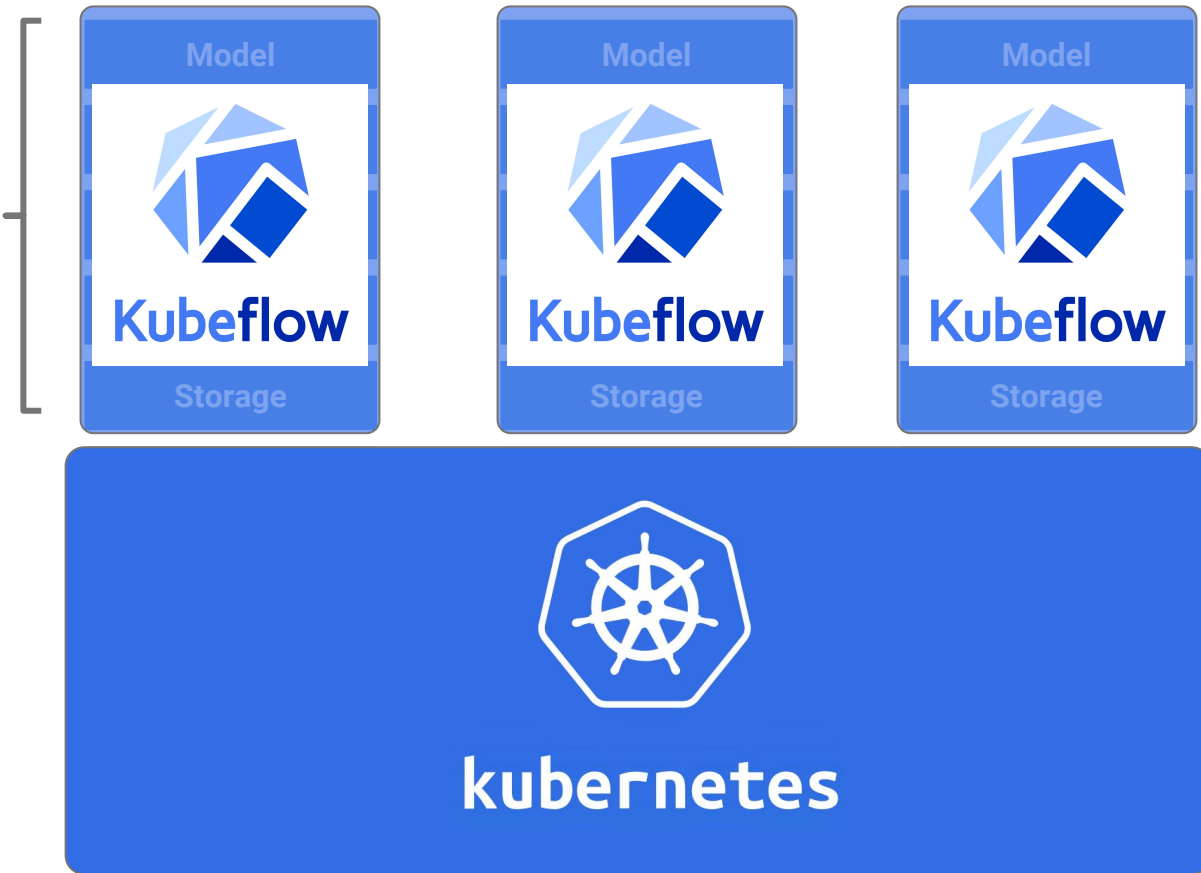


Portability

Laptop

Training Rig

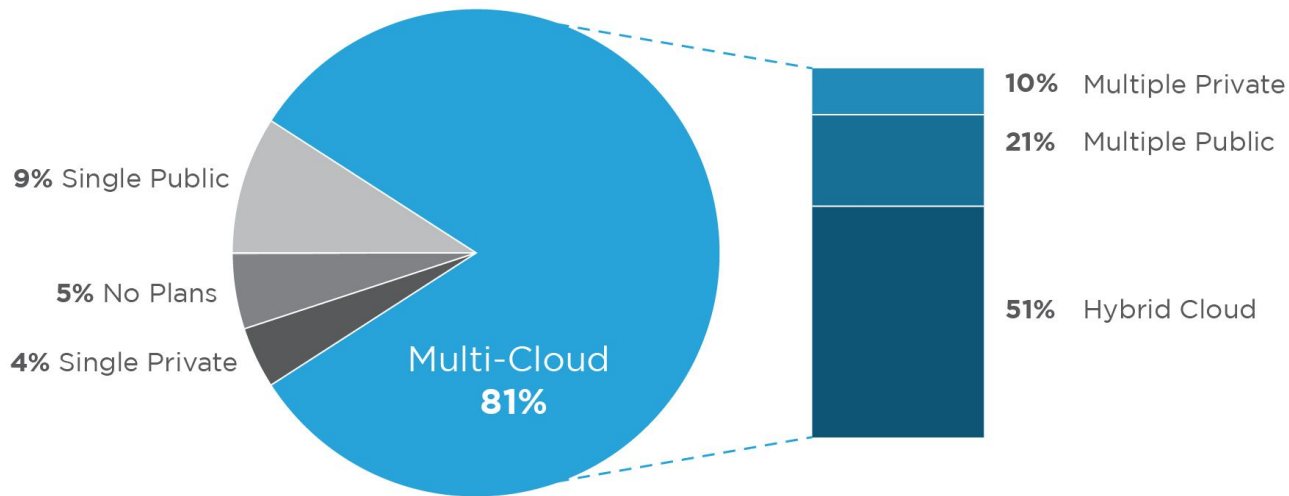
Cloud



Multi-Cloud is the Reality

Respondents with 1,000+ Employees

81% of enterprises have a multi-cloud strategy



And Not Just One Cloud!

Companies using almost **5** public
and private clouds on average

Public + Private Clouds Used	Average <i>All respondents</i>	Median <i>All respondents</i>
Running Applications	3.1	3.0
Experimenting	1.7	1.0
Total	4.8	4.0

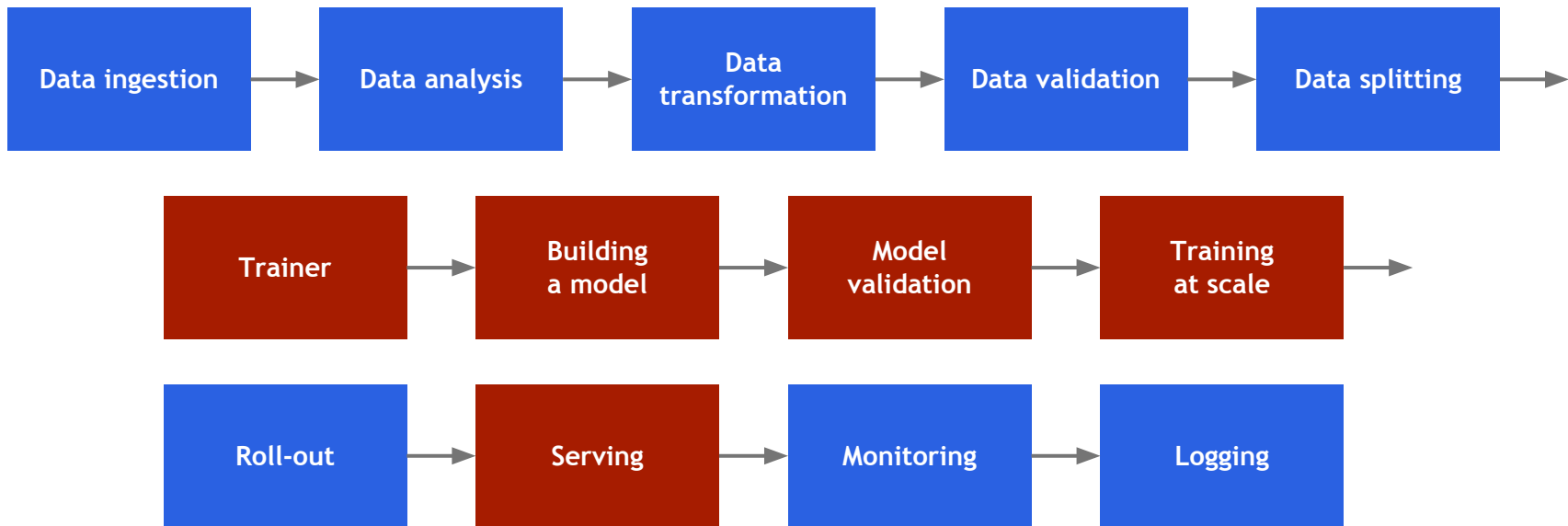
Scalability

- **More** accelerators (GPU, TPU)
- **More** CPUs
- **More** disk/networking
- **More** skillsets (SWEs, data scientists)
- **More** teams
- **More experiments**

What's in the box?

- **Jupyter notebook**
- Multi-architecture, **distributed training**
- Multi-framework, **model serving**
- Examples and walkthroughs for getting started
- **Ksonnet packaging for customizing it yourself!**

What's in the box?



Reminder: What you did NOT see:



Bespoke Solutions



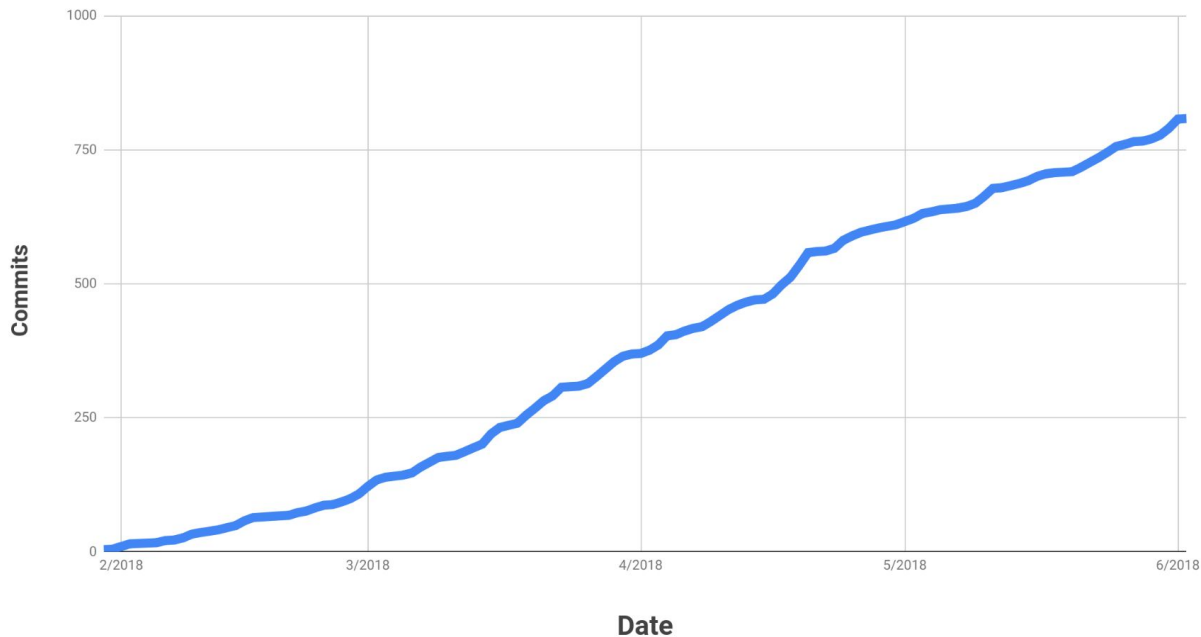
Cloud-specific, non-portable tech



Forking of Kubernetes APIs

Momentum!

Commits Since Launch



- 800+ commits
- 70+ Community contributors
- 17+ Companies contributing, including:



Kubeflow is Open!



Open
community



Open
design



Open
source



Open
to ideas

Kubeflow Components

Bootstrap Container

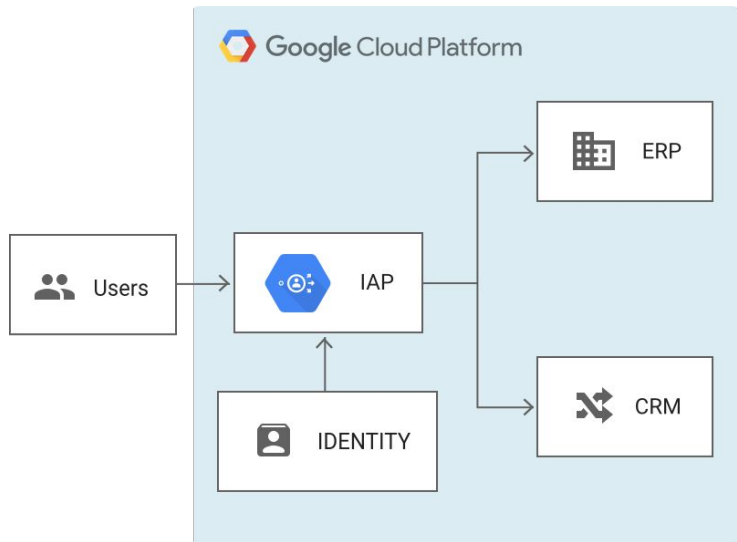
- Bootstrap is a tool to manage ksonnet application configured to take advantage of a user's cluster
- It lives in a Kubernetes container
- It deploys every other Kubeflow component
- The tool collects information from a variety of sources:
 - Kubeconfig file
 - K8s master
 - User input
 - Cloud environment
- ...and based on the results chooses good values for various Kubeflow parameters.

Ambassador Service

- Ambassador provides all the functionality of a traditional ingress controller (i.e., path-based routing) while exposing many additional capabilities such as authentication, URL rewriting, CORS, rate limiting, and automatic metrics collection
- Ambassador is designed to allow service authors to control how their service is published
- Ambassador is deployed as a Kubernetes service of type **LoadBalancer**.
- More: <https://www.getambassador.io/user-guide/getting-started>

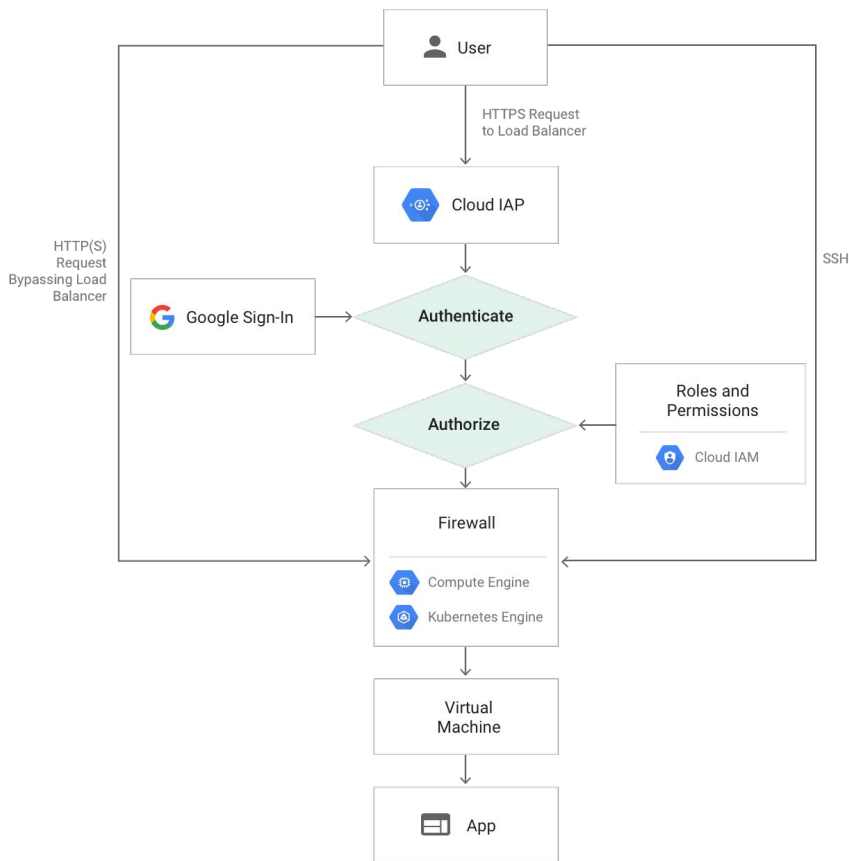
Cloud IAP

- Cloud IAP lets you establish a central authorization layer for applications accessed by HTTPS, so you can use an application-level access control model instead of relying on network-level firewalls.
- How Cloud IAP works:
<https://cloud.google.com/iap/docs/concepts-overview>
- IAP in K8s:
<https://cloud.google.com/iap/docs/enabling-kubernetes-howto>



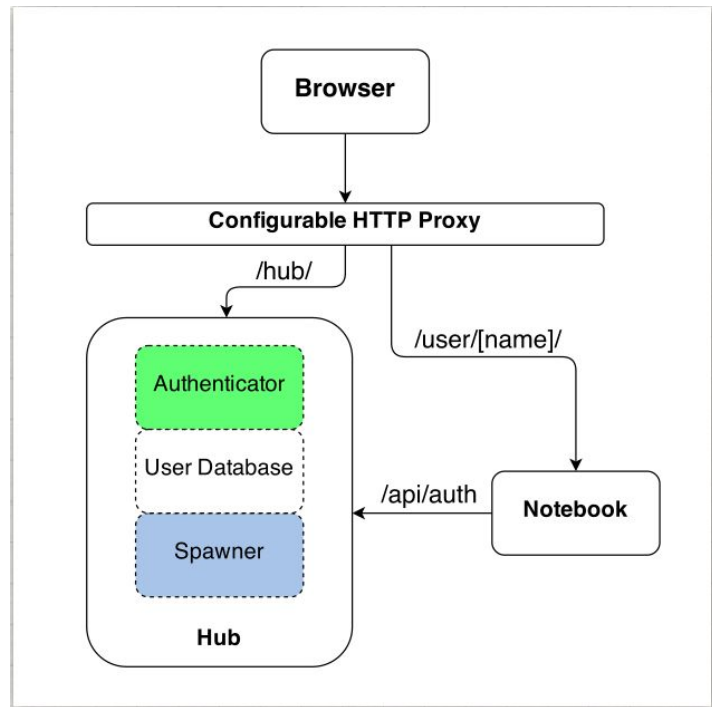
Cloud IAP

- When an application or resource is protected by Cloud IAP, it can only be accessed through the proxy by members, also known as users, who have the correct Cloud Identity and Access Management (Cloud IAM) role.
- When you grant a user access to an application or resource by Cloud IAP, they're subject to the fine-grained access controls implemented by the product in use without requiring a VPN.
- When a user tries to access a Cloud IAP-secured resource, Cloud IAP performs authentication and authorization checks.



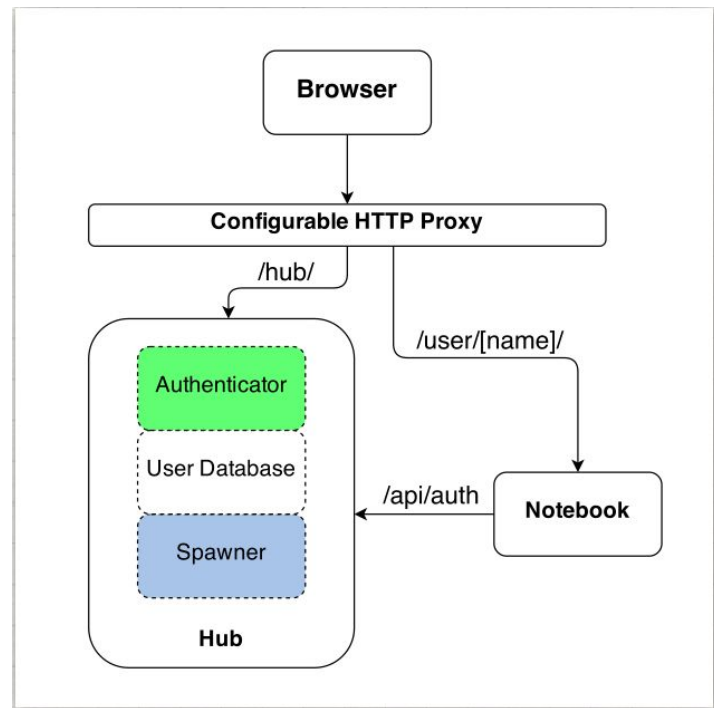
Jupyter Service and Hub (JH)

- With JupyterHub you can create a **multi-user Hub** which spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server.
- Project Jupyter created JupyterHub to support many users. The Hub can offer **notebook servers** to a **class of students**, a corporate **data science workgroup**, a **scientific research project**, or a high performance computing group



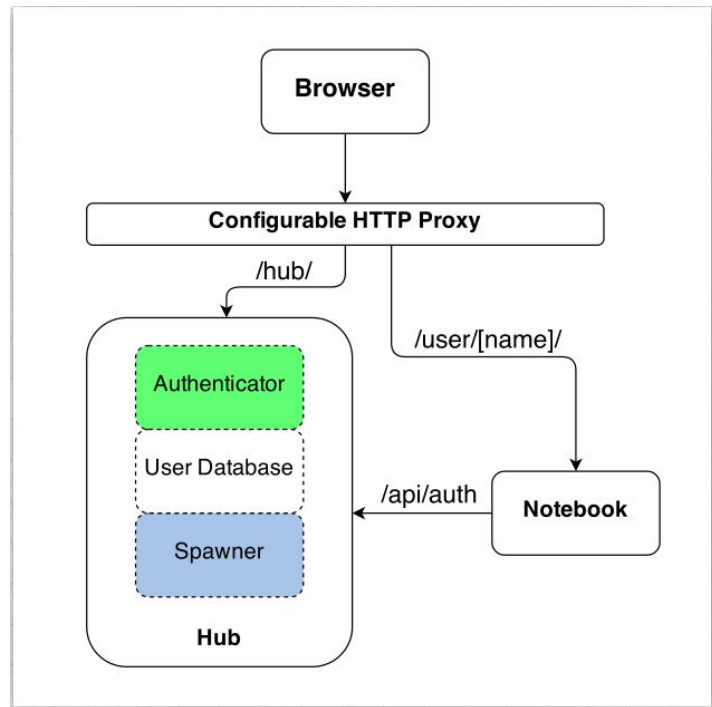
Jupyter Service and Hub (JH)

- Three subsystems make up JupyterHub:
 - a multi-user Hub (tornado process)
 - a configurable http proxy (node-http-proxy)
 - multiple single-user Jupyter notebook servers (Python/IPython/tornado)



Jupyter Service and Hub (JH)

- JupyterHub performs the following functions:
 - The Hub launches a proxy
 - The proxy forwards all requests to the Hub by default
 - The Hub handles user login and spawns single-user servers on demand
 - The Hub configures the proxy to forward URL prefixes to the single-user notebook servers



Jupyter Service and Hub (JH)

- The JupyterHub notebooks include:
 - [TFT](#) - Tensorflow Transform
 - a library for preprocessing data with TensorFlow
 - [TFMA](#) - Tensorflow Model Analysis
 - a library for evaluating TensorFlow models. It allows users to evaluate their models on large amounts of data in a distributed manner, using the same metrics defined in their trainer. These metrics can be computed over different slices of data and visualized in Jupyter notebooks

Jupyter - For the Data Scientist

Before Kubeflow:

- `curl -O https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86_64.sh`
- `bash -c Anaconda3-5.0.1-Linux-x86_64.sh`
- `<several license/etc steps>`
- `conda create -y -n mlenv python=2 pip scipy gevent sympy`
- `source activate mlenv`
- `pip install tensorflow==1.7.0 | tensorflow-gpu==1.7.0`
- `open http://127.0.0.1:8080`

With Kubeflow (either UI or CLI supported):

- `curl <SDK-URL> | gunzip; ./google-cloud-sdk/install.sh`
- `gcloud components install kubectl`
- `git clone git@github.com:kubeflow/kubeflow.git`
- `vi gke/configs/cluster-kubeflow.yaml # Set for project specific settings`
- `vi gke/configs/env-kubeflow.yaml # Set for project specific settings`
- `./gke/configs/deploy.sh`
- `wget <bootstrap.yml-url> | kubectl create -f`
- `open http://kubeflow.endpoints.MY_PROJECT_NAME.cloud.goog`

gcloud
Setup



K8s
Setup



With Kubeflow 1-click Deployment (either UI or CLI supported):

- `curl <SDK-URL> | gunzip; ./google-cloud-sdk/install.sh`
- `gcloud deployment-manager deployments create kubeflow-deployment --config vm.yaml`
- `open http://kubeflow.endpoints.MY_PROJECT_NAME.cloud.goog`

Jupyter - For the Data Scientist

With Minikube-based (local) Kubeflow on OS X:

- < Install VirtualBox >
- `curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.28.0/minikube-darwin-amd64`
- `chmod +x minikube; sudo mv minikube /usr/local/bin/`
- `brew install kubectl`
- # Below inset because it's just alias setting
 - `POD=`kubectl -n kubeflow get pods --selector=service=ambassador | awk '{print $1}' | tail -1``
 - `kubectl -n kubeflow port-forward $POD 8080:80 2>&1 >/dev/null &`
 - `POD=`kubectl -n kubeflow get pods --selector=app=tf-hub | awk '{print $1}' | tail -1``
 - `kubectl -n kubeflow port-forward $POD 8000:8000 2>&1 >/dev/null &`
- `wget <bootstrap.yml-url> | kubectl create -f`
- `open http://localhost:8000`



K8s
Setup

Jupyter - For the Data Scientist

- Why bother with all the extra layers?
 - Same environment in development, staging and production
 - Distributed training and/or TPUs
 - Enables use of downloadable, sophisticated pipelines for accelerating usage (not just hello-world.ipyp)

TFJob

- TFJob provides a Kubernetes **custom resource** that makes it easy to run distributed or non-distributed TensorFlow jobs on Kubernetes.
- Using a **Custom Resource Definition** (CRD) gives users the ability to create and manage TF Jobs just like builtin K8s resources.
- For example to create a job:
 - `kubectl create -f examples/tf_job.yaml`
- To list jobs:
 - `kubectl get tfjobs`

■	NAME	KINDS
■	example-job	TFJob.v1alpha.kubeflow.org

TFJob

- The **TFJob CRD** defines a TFJob resource for K8s.
- The **TFJob** resource is a collection of **TfReplicas**.
- Each TfReplica corresponds to a set of TensorFlow processes performing a role in the job; e.g. master, parameter server or worker.

```

apiVersion: "kubeflow.org/v1alpha1"
kind: "TFJob"
metadata:
  name: "example-job"
spec:
  replicaSpecs:
    - replicas: 1
      tfReplicaType: MASTER
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              args:
                - --log_dir=gs://my-job/log-dir
              restartPolicy: OnFailure
    - replicas: 2
      tfReplicaType: WORKER
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              args:
                - --log_dir=gs://my-job/log-dir
              restartPolicy: OnFailure
    - replicas: 1
      tfReplicaType: PS

```

An example job spec for a distributed Training job with 1 master, 2 workers and 1 PS. (Parameter Server Task)

TFJob

- TFJob requires three components:
 - Master
 - Parameter servers
 - Workers

- More:

https://github.com/kubeflow/tf-operator/blob/master/tf_job_design_doc.md

```

apiVersion: "kubeflow.org/v1alpha1"
kind: "TFJob"
metadata:
  name: "example-job"
spec:
  replicaSpecs:
    - replicas: 1
      tfReplicaType: MASTER
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              args:
                - --log_dir=gs://my-job/log-dir
              restartPolicy: OnFailure
    - replicas: 2
      tfReplicaType: WORKER
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              args:
                - --log_dir=gs://my-job/log-dir
              restartPolicy: OnFailure
    - replicas: 1
      tfReplicaType: PS

```

Native Object

Master

Workers

Param Server

An example job spec for a distributed Training job with 1 master, 2 workers and 1 PS. (Parameter Server Task)

ML Pipeline Builder

- The ML Pipeline builder requires several CRDs. These include:
 - **Argo** CRD
 - **Scheduling** CRD
 - **Argo eventing** CRD
- Additionally, additional servers are added as well. These include:
 - **Pipeline DB**
 - **Pipeline web server** for front end UI
 - **Pipeline API server** to serve the list of the pipelines from the pipeline DB for persistence.

Argo

- Argo is an open source container-native **workflow engine** for getting work done on Kubernetes. Argo is **implemented as a Kubernetes CRD**
- With Argo:
 - **Define workflows** where each step in the workflow is a container.
 - Model multi-step workflows as a sequence of tasks or capture the dependencies between tasks **using a graph (DAG)**.
 - Easily **run compute intensive jobs for machine learning** or data processing in a fraction of the time using Argo workflows on Kubernetes.
 - **Run CI/CD pipelines** natively on Kubernetes without configuring complex software development products.

Argo and Pipeline Builder

- Currently, Argo is **very** verbose for its workflow construction
- We have a new project “ML Pipeline Builder” that will help solve this with a DSL (Argo's YAML Domain Specific Language)
- This will enable many frequently used scenarios such as scheduling and triggering.
- Additionally, this will unlock component reuse by SWEs.

```
def video_recommender_pipeline(videos_path, ratings_path, ...):
    processed_vids = process_raw_videos(<path/uri to container>, videos_path, ...)
    labelled_data_path = merge_with_ratings(<path/uri to container>, ratings_path, ...)
    raw_features = compute_features(<path/uri to container>, labelled_data_path, ...)

    transformed_features = tf_transform(labelled_data_path, ...)
    train, test, val = split_data(feature_vecs, ...)
    model = hypertune(custom_tf_train(<path/uri to container>, train, val, ...))
    result = tf_model_analysis(model, ...)

    deploy_online_serving(model, result, ...)
```

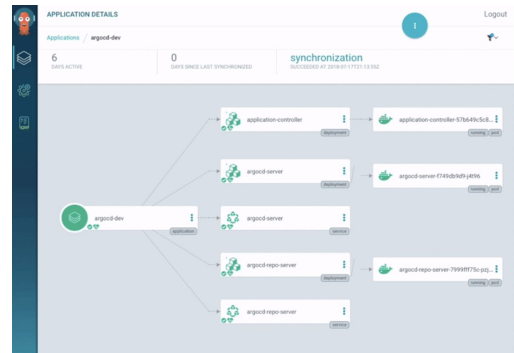
Pipeline
Builder

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: xgb-
spec:
  arguments:
    parameters:
      - name: project
      - name: clustername
      - name: output
      - name: train-data
      - name: eval-data
      - name: train-conf
      - name: schema
  entrypoint: xgb
  templates:
    - dag:
        tasks:
          - arguments:
              parameters:
                - name: project
                  value: '{{inputs.parameters.project}}'
                - name: clustername
                  value: '{{inputs.parameters.clustername}}'
                - name: output
                  value: '{{inputs.parameters.output}}'
                - name: transform-eval
                  value: '{{tasks.transform.outputs.parameters.transform-eval}}'
                - name: target
                  value: '{{inputs.parameters.target}}'
                - name: analyze-output
                  value: '{{tasks.analyze.outputs.parameters.analyze-output}}'
                - name: train-model
                  value: '{{tasks.train.outputs.parameters.train-model}}'
            dependencies:
              - analyze
              - train
              - transform
            name: predict
            template: predict
          - arguments:
              parameters:
                - name: project
                  value: '{{inputs.parameters.project}}'
                - name: region
                  value: '{{inputs.parameters.region}}'
                - name: clustername
                  value: '{{inputs.parameters.clustername}}'
                - name: output
                  value: '{{inputs.parameters.output}}'
            dependencies: []
            name: create-cluster
            template: create-cluster
          - arguments:
              parameters:
                - name: project
                  value: '{{inputs.parameters.project}}'
```

Argo yaml output

Argo for CI/CD

- Argo CI is a continuous integration and deployment system powered by Argo workflow engine for Kubernetes. Argo CI provides integration with SCM (currently only Github is supported) and automatically triggers CI workflow defined using Argo YAML DSL.
- Argo CD follows the GitOps pattern of using git repositories as the source of truth for defining the desired application state. Kubernetes manifests can be specified in several ways:
 - ksonnet applications
 - helm charts
 - Simple directory of YAML/json manifests

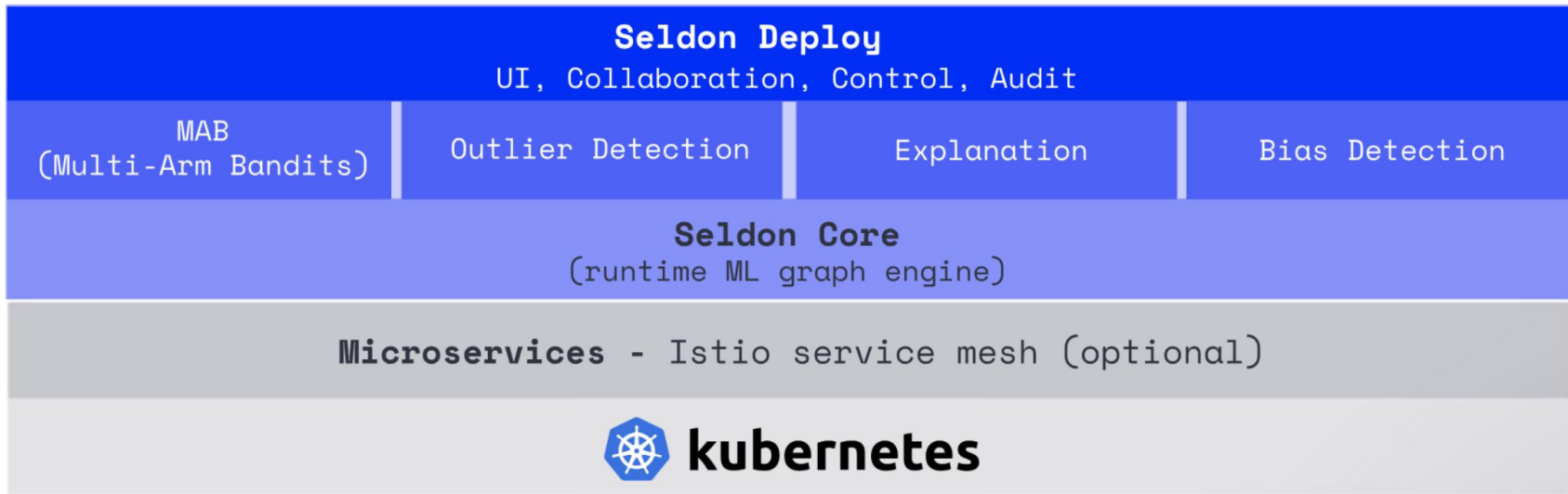


TFService

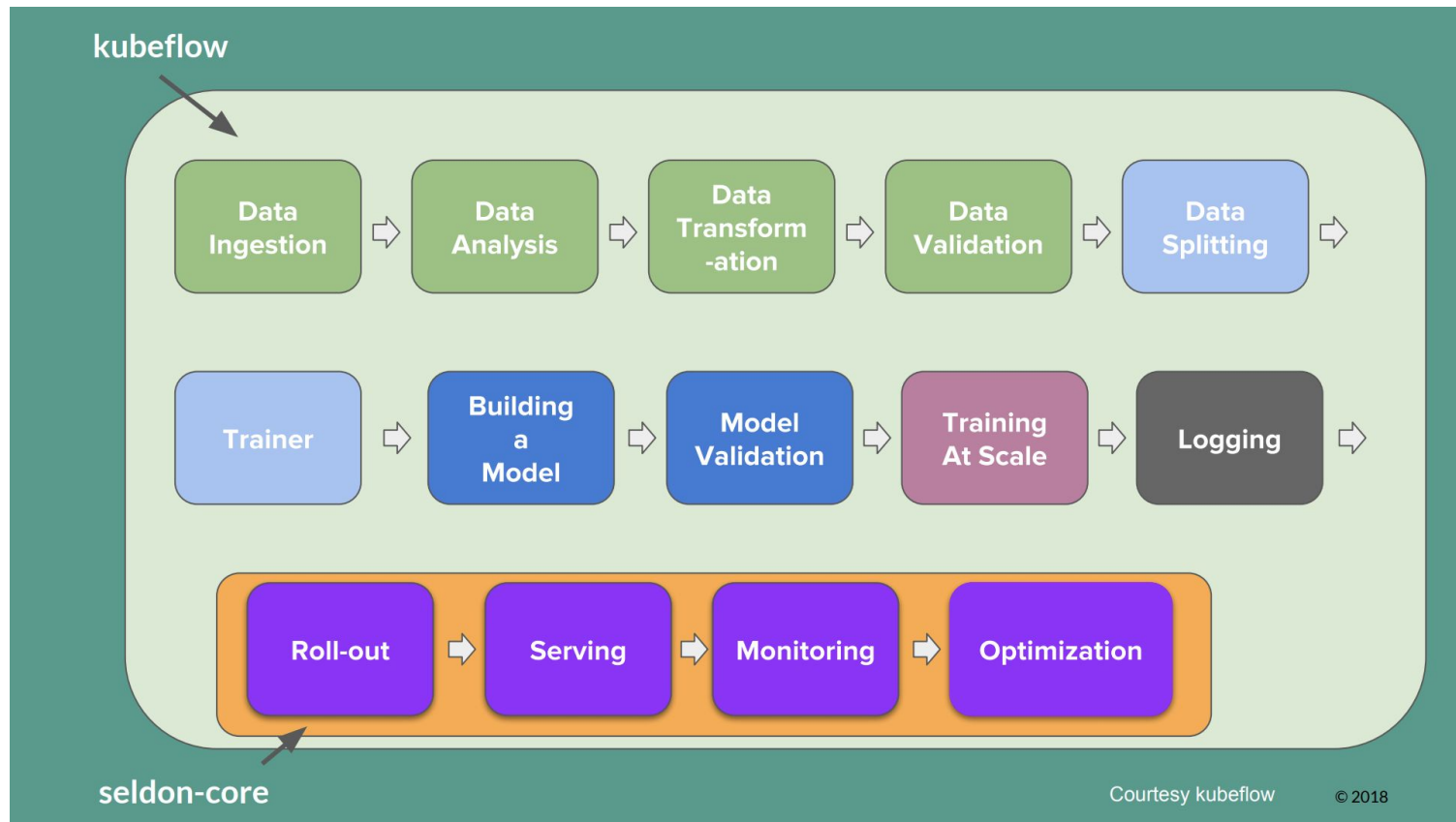
- The library files for **tf-serving** define a set of relevant parts (e.g., deployments, **services**, secrets, and so on) that can be combined to configure tf-serving for a wide variety of scenarios.
- It provides a set of pre-fabricated "flavors" (or "distributions") of tf-serving, each of which is configured for a different use case. These are captured as ksonnet prototypes, which allow users to interactively customize these distributions for their specific needs.

Seldon

- Seldon Core, our open-source framework, makes it easier and faster to deploy your machine learning models and experiments at scale on Kubernetes.
- Serve your models built in any open-source or commercial model building framework.
- Leverage powerful Kubernetes features like Custom Resource Definitions to manage model graphs. And connect your continuous integration and deployment (CI/CD) tools to scale and update your deployment.
- Can be run in Kubeflow as the Model Serving component.



Seldon-core

[Seldon Slides](#)

Kubeflow Architecture

Access Jupyter

One notebook please.



Laptop

kubeflow source

gcloud sdk kubectl

GKE

IAP

Ambassador Svc

Ambassador

Ambassador

Ambassador

JH Svc

JH Hub

TFT

TFMA

Bootstrap Container

ksonnet

Pipeline Svc

Pipeline UI

Pipeline API Server

Pipeline DB

K8s API

TF Job CRD

Pipeline CRDs



AI Hub



KF Registry



GCR



Block Store



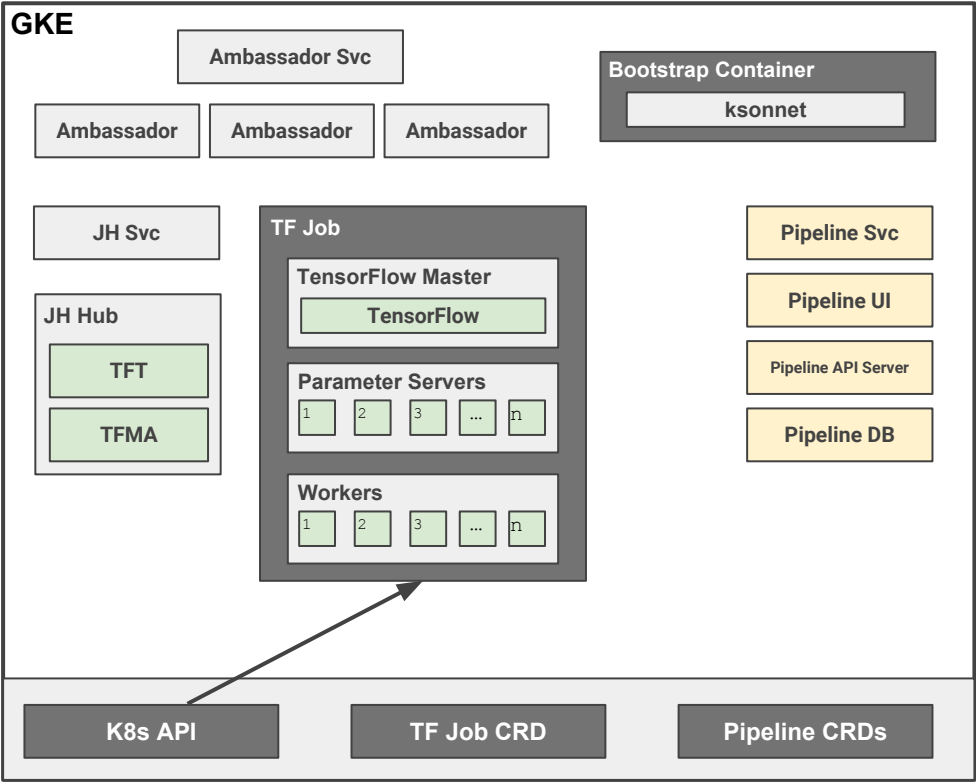
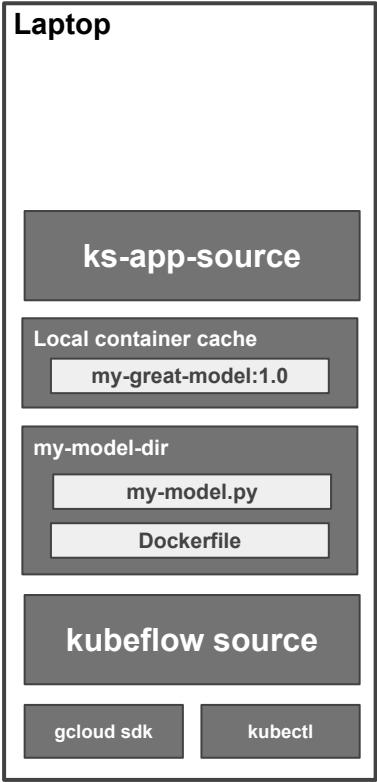
Data Source



GCP Control Plane

Build a TF Job

Finally, submit the job to the K8s master.



AI Hub



KF Registry



GCR



Block Store



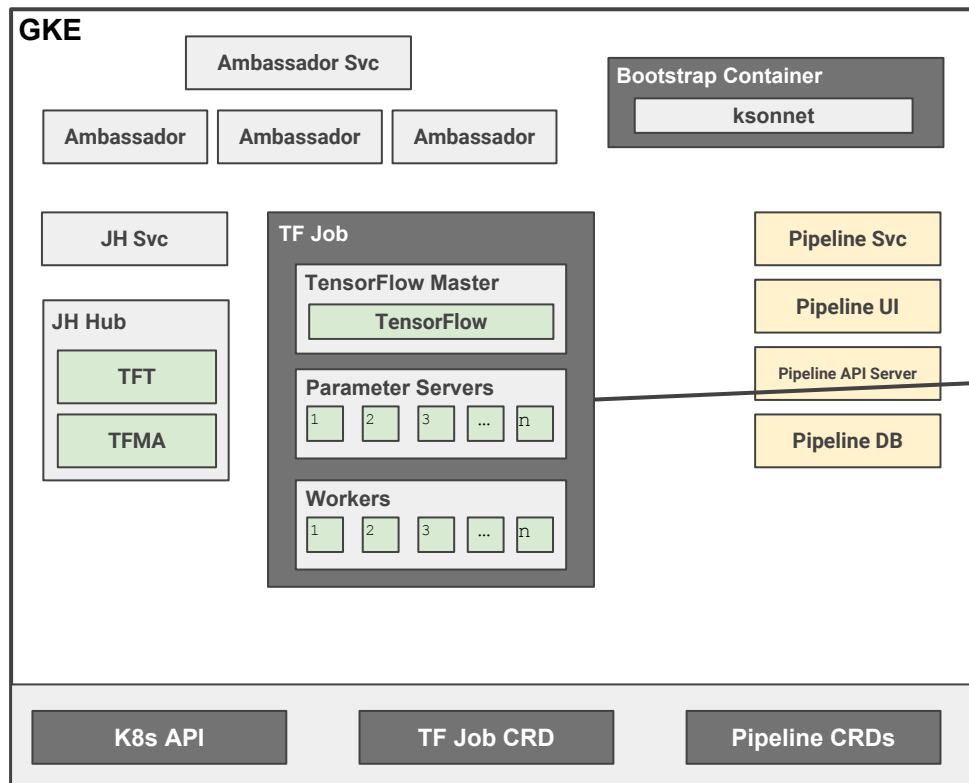
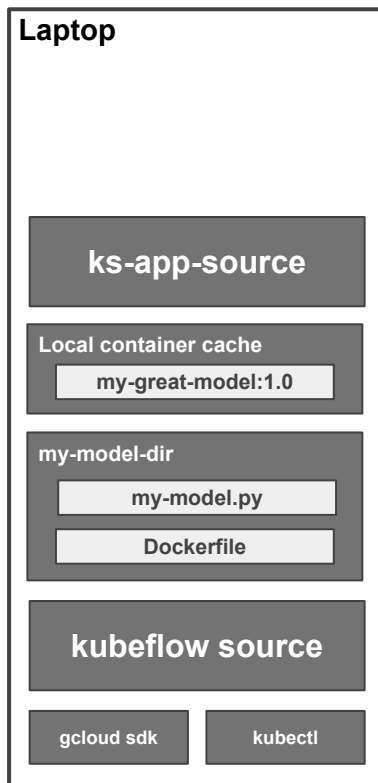
Data Source



GCP Control Plane

```
$ be ks apply kubeflow -c my-great-model-1_0-job
```

TF Training Job runs to completion, storing data on GCS; Job auto-deletes once done



AI Hub



KF Registry



GCR



Block Store



Data Source



GCP Control Plane

```
$ ks apply kubeflow -c my-great-model-1_0-job
```

Serve Model

User Request



Laptop

ks-app-source

Local container cache
my-great-model:1.0

my-model-dir
my-model.py
Dockerfile

kubeflow source

gcloud sdk kubectl

GKE

IAP

Ambassador Svc

Ambassador Ambassador Ambassador

JH Svc

JH Hub
TFT
TFMA

TF Service

TF Serving 1

Bootstrap Container
ksonnet

Pipeline Svc
Pipeline UI
Pipeline API Server
Pipeline DB

K8s API TF Job CRD Pipeline CRDs



AI Hub



KF Registry



GCR



Block Store



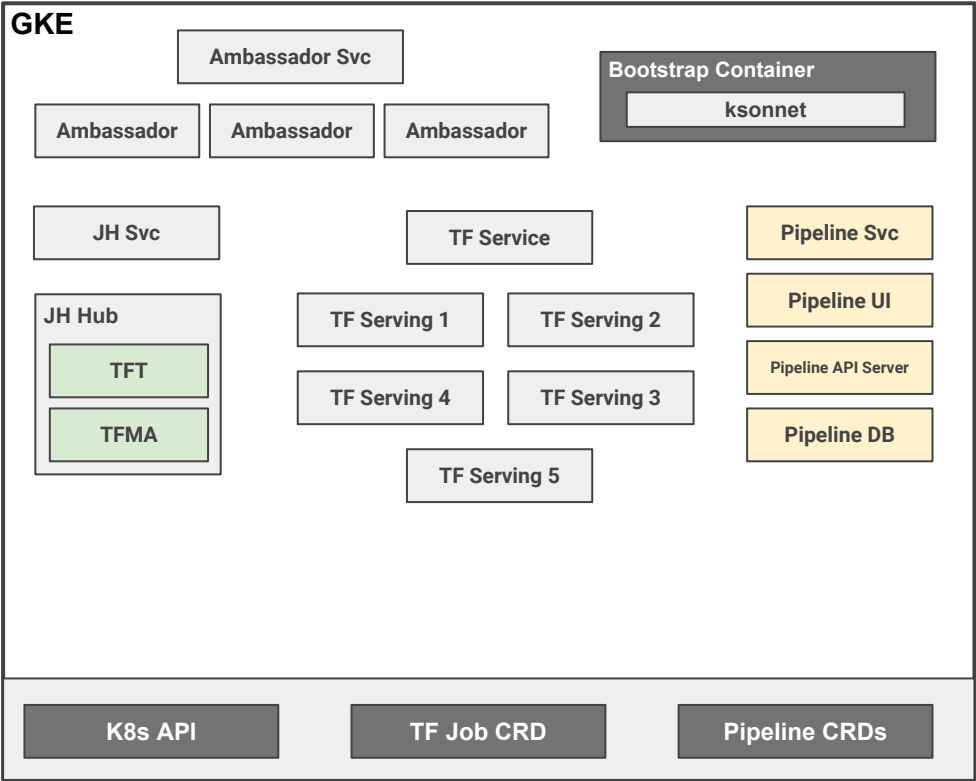
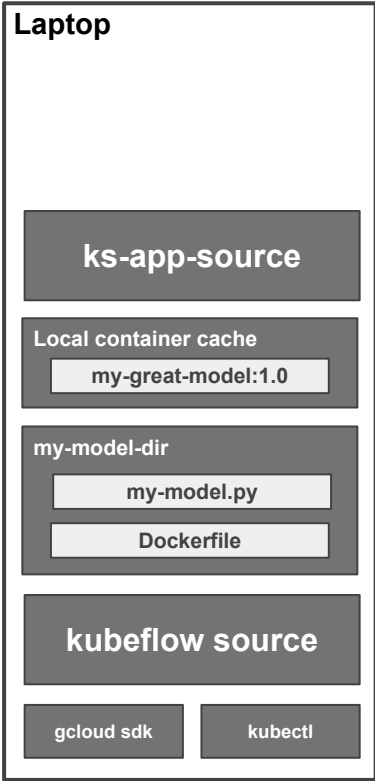
Data Source



GCP Control Plane

```
$ ks apply kubeflow -c serveMyGreatModelComponent
```

Scale Serving



AI Hub



KF Registry



GCR



Block Store



Data Source



GCP Control Plane

```
$ kubectl scale --replicas=5 deployment/serveMyGreatModel
```

Kubeflow

Tutorial

Setting up a GKE Cluster

- Install **gcloud**
 - `curl <SDK-URL> | gunzip; ./google-cloud-sdk/install.sh`
- Install **kubect**
 - `gcloud components install kubectl`
- Create **GKE Cluster** - Clone the repo so I can edit the creation files
 - `curl <SDK-URL> | gunzip; ./google-cloud-sdk/install.sh`
- Create **GKE Cluster** - Edit the cluster setup settings (e.g. # of nodes)
 - `vi gke/configs/cluster-kubeflow.yaml # Set for project specific settings`

cluster-kubeflow.yaml

- Deployment manager file details infrastructure specific settings for a cluster. E.g.
 - # of nodes
 - # of GPU nodes
 - Size of nodes
 - User name for admin
 - IP range
 - Etc.
- Uses .jinja templating and can be customized
- **./deploy.sh merges with env-kubeflow.yaml and executes against GCP**

```

1. bootstrapperConfig: |
2.   app:
3.     packages:
4.       - name: core
5.         registry: kubeflow
6.       - name: tf-serving
7.         registry: kubeflow
8.       - name: tf-job
9.         registry: kubeflow
10.      - name: pytorch-job
11.        registry: kubeflow
12.     components:
13.       - name: kubeflow-core
14.         prototype: kubeflow-core
15.       - name: cloud-endpoints
16.         prototype: cloud-endpoints
17.       - name: cert-manager
18.         prototype: cert-manager
19.       - name: iap-ingress
20.         prototype: iap-ingress
21.       - name: pytorch-operator
22.         prototype: pytorch-operator
23.     parameters:
24.       - component: cloud-endpoints
25.         name: secretName
26.         value: admin-gcp-sa
27.       - component: cert-manager
28.         name: acmeEmail
29.         # TODO: use your email for ssl cert
30.         value: johnDoe@acme.com
31.       - component: iap-ingress
32.         name: ipName
33.         # TODO: make sure value of ipName is the same as property <ipName>.
34.         value: ipName
35.       - component: iap-ingress
36.         name: hostname
37.         # TODO: replace with Name of GCP project. This is fully qualified domain name to use with ingress.
38.         value: kubeflow.endpoints.<Project>.cloud.goog
39.       - component: kubeflow-core
40.         name: jupyterHubAuthenticator
41.         value: iap

```

Setting up a GKE Cluster

- Pull in my **env variables**
 - `vi gke/configs/env-kubeflow.yaml # Set for project specific settings`

env-kubeflow.sh

- Sub script executed by deploy.sh
- Pulls in environment variables that are used during setup

```
1. #!/usr/bin/env bash
2. #
3. # Script that defines various environment variables.
4. # This is script defines values for all the variables used in
5. # the instructions.
6.
7. # Bucket and project must be unique for each project
8.
9. # Set PROJECT to the project you want to use with Kubeflow.
10. export PROJECT=<your_project>
11.
12. # Set DEPLOYMENT_NAME to the name to give to the deployment.
13. # The name must be unique for each deployment within your project.
14. export DEPLOYMENT_NAME=kubeflow
15.
16. # Set this to the zone in your ${CONFIG_FILE}
17. export ZONE=us-east1-d
18.
19. # Set config file to the YAML file defining your deployment manager configs.
20. export CONFIG_FILE=cluster-kubeflow.yaml
```

Setting up a GKE Cluster

- Execute **deploy.sh** to create cluster
 - `./gke/configs/deploy.sh`

deploy.sh

- Executes a series of gcloud commands to set up an appropriate GKE cluster
- Merges with environment variables and cluster-kubeflow.yaml for user specific settings
- Includes service Utility script to can be used to deploy Kubeflow end-to-end.
- Ensures appropriate gcloud APIs are enabled
- Uses gcloud deployment manager
- Currently also sets up kubeflow (this functionality may be split out now that we have the bootstrap container)

```
1. # Required Variables
2. export PROJECT=${PROJECT:-}
3. export DEPLOYMENT_NAME=${DEPLOYMENT_NAME:-}
4. export ZONE=${ZONE:-}
5. export CONFIG_FILE=${CONFIG_FILE:-}
6. export CLIENT_ID=${CLIENT_ID:-}
7. export CLIENT_SECRET=${CLIENT_SECRET:-}
8.
9. if [ -z "${PROJECT}" ] || \
10. [ -z "${DEPLOYMENT_NAME}" ] || \
11. [ -z "${ZONE}" ] || \
12. [ -z "${CONFIG_FILE}" ] || \
13. [ -z "${CLIENT_ID}" ] || \
14. [ -z "${CLIENT_SECRET}" ]; then
15. echo 'Required variables missing. Please check again!'
16. exit 1
17. fi
18.
19. if [[ ! -f "${CONFIG_FILE}" ]]; then
20. echo "Config file ${CONFIG_FILE} does not exist!"
21. exit 1
22. fi
23.
24. # Computed Variables
25. export PROJECT_NUMBER=$(gcloud projects describe ${PROJECT} --format='value(project_number)')
26. export SA_EMAIL=${DEPLOYMENT_NAME}-admin@${PROJECT}.iam.gserviceaccount.com
27. export USER_EMAIL=${DEPLOYMENT_NAME}-user@${PROJECT}.iam.gserviceaccount.com
28. export USER_SECRET_NAME=${DEPLOYMENT_NAME}-user
29. export K8S_ADMIN_NAMESPACE=kubeflow-admin
30. export K8S_NAMESPACE=kubeflow
31.
32. # Enable GCloud APIs
33. gcloud services enable deploymentmanager.googleapis.com --project=${PROJECT}
34. gcloud services enable servicemanagement.googleapis.com --project=${PROJECT}
35. gcloud services enable iam.googleapis.com --project=${PROJECT}
36.
37. # Set IAM Admin Policy
38. gcloud projects add-iam-policy-binding ${PROJECT} \
39. --member serviceAccount:${PROJECT_NUMBER}@cloudservices.gserviceaccount.com \
40. --role roles/resourcemanager.projectIamAdmin
41.
42. # Run Deployment Manager
43. gcloud deployment-manager --project=${PROJECT} deployments create ${DEPLOYMENT_NAME} --config=${CONFIG_FILE}
44.
45. # TODO(jlewi): We should name the secrets more consistently based on the service account name.
46. # We will need to update the component configs though
47. gcloud --project=${PROJECT} iam service-accounts keys create ${SA_EMAIL}.json --iam-account ${SA_EMAIL}
48. gcloud --project=${PROJECT} iam service-accounts keys create ${USER_EMAIL}.json --iam-account ${USER_EMAIL}
49.
50. # Set credentials for kubectl context
51. gcloud --project=${PROJECT} container clusters get-credentials --zone=${ZONE} ${DEPLOYMENT_NAME}
52.
53. # Ignore errors from now onwards. If secret/namespace already exists just keep going.
54. set +e
55.
56. # The namespace kubeflow may not exist yet because the bootstrapper can't run until the admin-gcp-sa
57. # secret is created.
58. kubectl create namespace ${K8S_NAMESPACE}
59.
60. # We want the secret name to be the same by default for all clusters so that users don't have to set it manually.
61. kubectl create secret generic --namespace=${K8S_ADMIN_NAMESPACE} admin-gcp-sa --from-file=admin-gcp-sa.json=./${SA_EMAIL}.json
62. kubectl create secret generic --namespace=${K8S_NAMESPACE} admin-gcp-sa --from-file=admin-gcp-sa.json=./${SA_EMAIL}.json
63. kubectl create secret generic --namespace=${K8S_NAMESPACE} user-gcp-sa --from-file=user-gcp-sa.json=./${USER_EMAIL}.json
```

deploy.sh

Bootstrap Kubeflow

- Bootstrap quick start:
 - `kubectl create -f bootstrapper.yaml`
- After running that command, you should have kubeflow components deployed inside your k8s cluster.
- The default components are defined in default.yaml
- Users can customize which components to deploy by pointing `--config` args in `bootstrapper.yaml` to their own config (eg. a configmap in k8s cluster)

bootstrapper.yaml

- In order to run a job on Kubernetes, a user needs to hand the K8s API a manifest for how to run
- The bootstrapper.yaml file creates all the necessary components for the bootstrap container to run on K8s.
- Includes:
 - Namespace
 - RBAC
 - PVC
 - Statefulset

```
1  ---
2  # Namespace for bootstrapper
3  apiVersion: v1
4  kind: Namespace
5  metadata:
6    name: kubeflow-admin
7  ---
8  # Make kubeflow-admin admin
9  apiVersion: rbac.authorization.k8s.io/v1beta1
10 kind: ClusterRoleBinding
11 metadata:
12   name: kubeflow-cluster-admin
13 subjects:
14   - kind: ServiceAccount
15     name: default
16     namespace: kubeflow-admin
17 roleRef:
18   kind: ClusterRole
19   name: cluster-admin
20   apiGroup: rbac.authorization.k8s.io
21 ---
22 # Store ksonnet apps
23 apiVersion: v1
24 kind: PersistentVolumeClaim
25 metadata:
26   name: kubeflow-ksonnet-pvc
27   namespace: kubeflow-admin
28   labels:
29     app: kubeflow-ksonnet
30 spec:
31   accessModes:
32     - ReadWriteOnce
33   resources:
34     requests:
35       storage: 5Gi
36   ---
37   apiVersion: apps/v1beta2
38   kind: StatefulSet
39   metadata:
40     name: kubeflow-bootstrapper
41     namespace: kubeflow-admin
42   spec:
43     selector:
44       matchLabels:
```

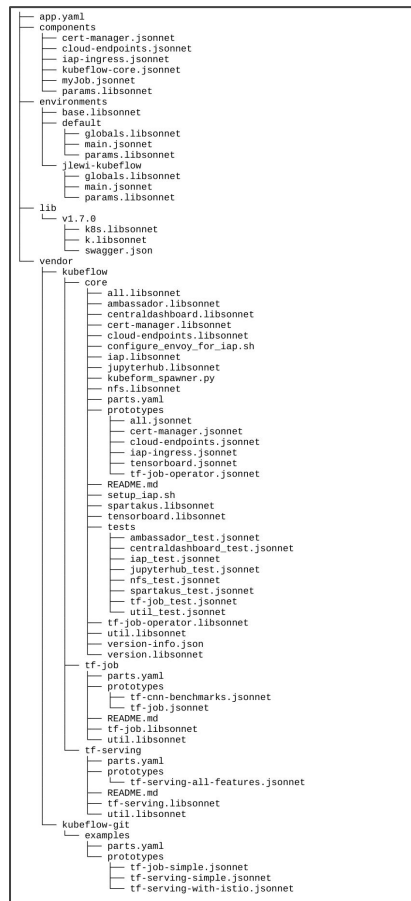
manifest for the bootstrap container

The Bootstrap Container

- Contains ksonnet and all necessary components to install Kubeflow packages
- Uses a simple yaml for detailing all components necessary

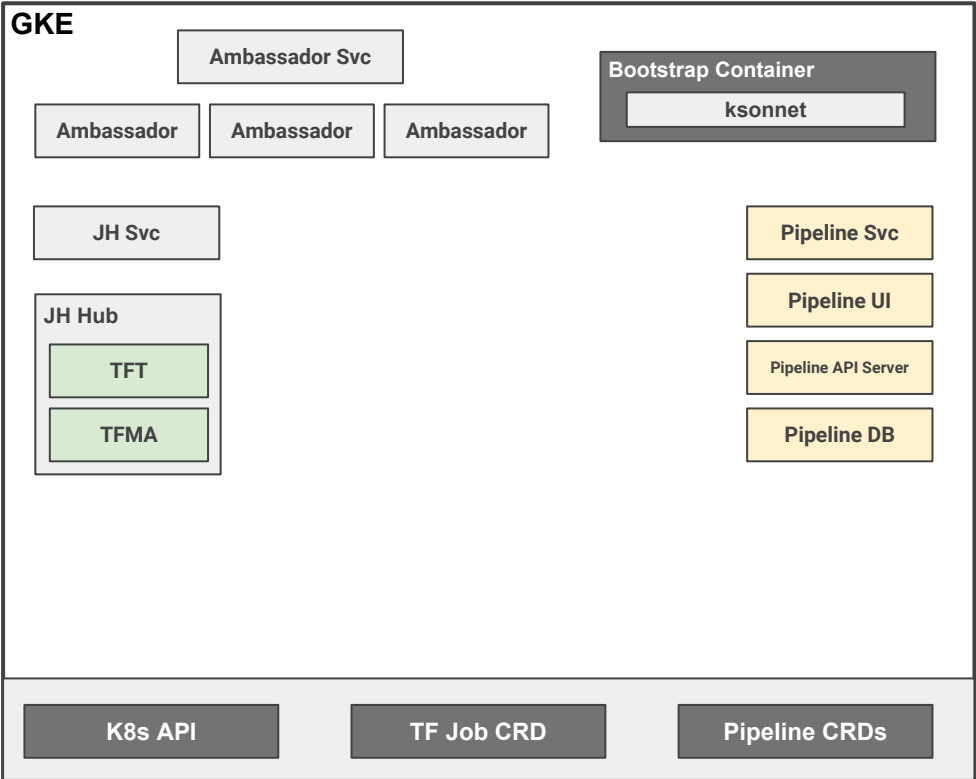
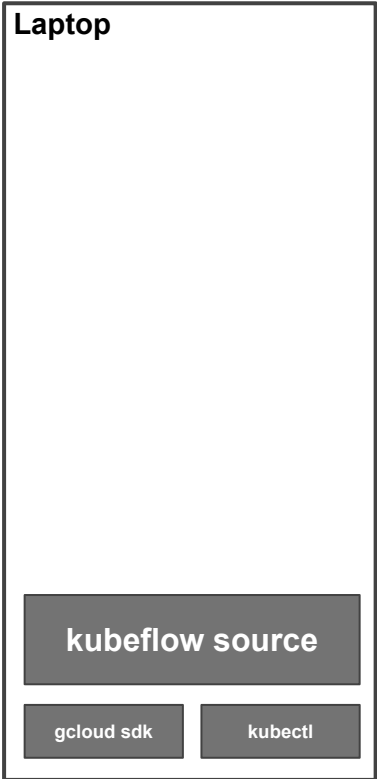
```

1 # Sample config for kubeflow bootstrapper
2 ---
3 # App only apply if on GKE
4 app:
5   packages:
6     - name: core
7       registry: kubeflow
8     - name: tf-serving
9       registry: kubeflow
10    - name: tf-job
11      registry: kubeflow
12  components:
13    - name: kubeflow-core
14      prototype: kubeflow-core
15    - name: cloud-endpoints
16      prototype: cloud-endpoints
17    - name: cert-manager
18      prototype: cert-manager
19    - name: iap-ingress
20      prototype: iap-ingress
21  parameters:
22    - component: cloud-endpoints
23      name: secretName
24      value: admin-gcp-sa
25    - component: cert-manager
26      name: acmeEmail
27      # TODO: use your email for ssl cert
28      value: johnDoe@acme.com
29    - component: iap-ingress
30      name: ipName
31      # TODO: make sure value of ipName is the same as property <ipName>.
32      value: ipName
33    - component: iap-ingress
34      name: hostname
35      # TODO: replace with Name of GCP project. This is fully qualified domain name to use with ingress.
36      value: kubeflow.endpoints.<Project>.cloud.goog
37    - component: kubeflow-core
38      name: jupyterHubAuthenticator
39      value: iap
  
```



Directory
structure of
installed
packages

Bootstraper - Creates all Kubeflow components



```
$ wget <bootstrap.yml-url> | kubectl create -f
```



AI Hub



KF Registry



GCR



Block Store



Data Source



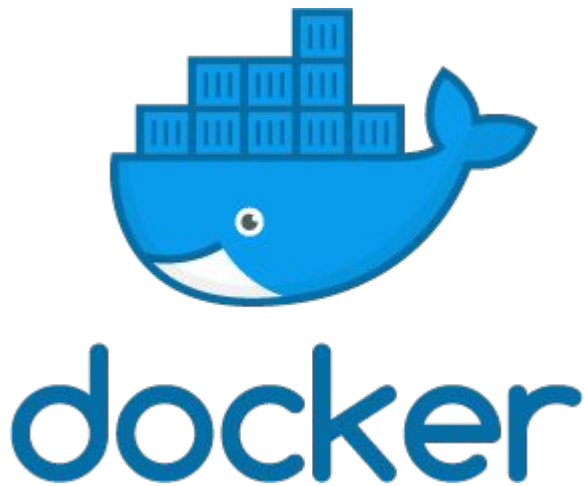
GCP Control Plane

Training - Steps

1. Install docker locally
2. Build container
3. Move container to GCR
4. Run TF Job using hosted container
 - a. Data gets stored on GCS
5. Start TF Serving pointing at data hosted on GCS

Training - 1. Install Docker Locally

- Go get docker and install it
 - <https://docs.docker.com/install/>



Training - 2. Build Container

- Usually the Dockerfile is very basic:

```
1  # Copyright 2016 The TensorFlow Authors. All Rights Reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #     http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 FROM tensorflow/tensorflow:1.7.0
16
17 ADD . /var/my-great-model
18 ENTRYPOINT ["python", "/var/my-great-model/my-great-model.py"]
```

Your Model



Training - 3. Move container to GCR

- By running, for example:
 - `docker push gcr.io/PROJECT-ID/my-great-model:1.0`

Training - 4. Run TF Job using hosted container

- Best practice is to make config changes locally, push to git, and then pull them in the bootstrap container.
- To set this up, it's a few one-time steps:
 - `be='kubectl exec bootstrap-container --namespace=kubeflow-admin'`
 - `be gcloud auth activate-service-account`
`--key-file=${GOOGLE_APPLICATION_CREDENTIALS}`
 - `be gcloud source repos create my-ks-app-repo`
 - `be gcloud source repos clone my-ks-app-repo git_my-ks-app-repo`
 - `be cp -r /opt/bootstrap/default ./git_my-ks-app-repo/ks-app`
 - `be git add ~/git_my-ks-app-repo/ks-app`
 - `be git commit -m "Kubeflow app"`
 - `be git push --set-upstream origin master`
- Then you clone this repo locally to make changes and push into your container when ready.
 - `git clone git_my-ks-app-repo/ks-app`

Training - 4. Run TF Job using hosted container

- Best practice is to make config changes locally, push to git, and then pull them in the bootstrap container.
- To set this up, it's a few one-time steps:
 - `be='kubectl exec bootstrap-container --namespace=kubeflow-admin'`
 - `be gcloud auth activate-service-account`
`--key-file=${GOOGLE_APPLICATION_CREDENTIALS}`
 - `be gcloud source repos create my-ks-app-repo`
 - `be gcloud source repos clone my-ks-app-repo git_my-ks-app-repo`
 - `be cp -r /opt/bootstrap/default ./git_my-ks-app-repo/ks-app`
 - `be git add ~/git_my-ks-app-repo/ks-app`
 - `be git commit -m "Kubeflow app"`
 - `be git push --set-upstream origin master`
- Then you clone this repo locally to make changes and push into your container when ready.
 - `git clone git_my-ks-app-repo/ks-app`

Training - 4. Run TF Job using hosted container

- Set the TF Job parameters in the bootstrap container
 - `be ks param set my-great-model-1_0-job image cr.io/PROJECT-ID/my-great-model:1.0`
- Generate the TF Job
 - `be ks generate tf-job my-great-model-1_0-job`
- And commit it to your repo
 - `be 'git add .; git commit -a -m "tf-job generated"; git push'`
- Finally, submit the job to the K8s master.
 - `be ks apply kubeflow -c my-great-model-1_0-job`

TF Job runs to completion, storing data on GCS; Job auto-deletes once done



Laptop

ks-app-source

Local container cache
my-great-model:1.0

my-model-dir
my-model.py
Dockerfile

kubeflow source

gcloud sdk kubectl

IAP

GKE

Ambassador Svc
Ambassador Ambassador Ambassador

JH Svc

JH Hub
TFT
TFMA

TF Job
TensorFlow Master
TensorFlow
Parameter Servers
1 2 3 ... n
Workers
1 2 3 ... n

Bootstrap Container
ksonnet

Pipeline Svc
Pipeline UI
Pipeline API Server
Pipeline DB

K8s API TF Job CRD Pipeline CRDs



AI Hub



KF Registry



GCR



Block Store



Data Source



GCP Control Plane

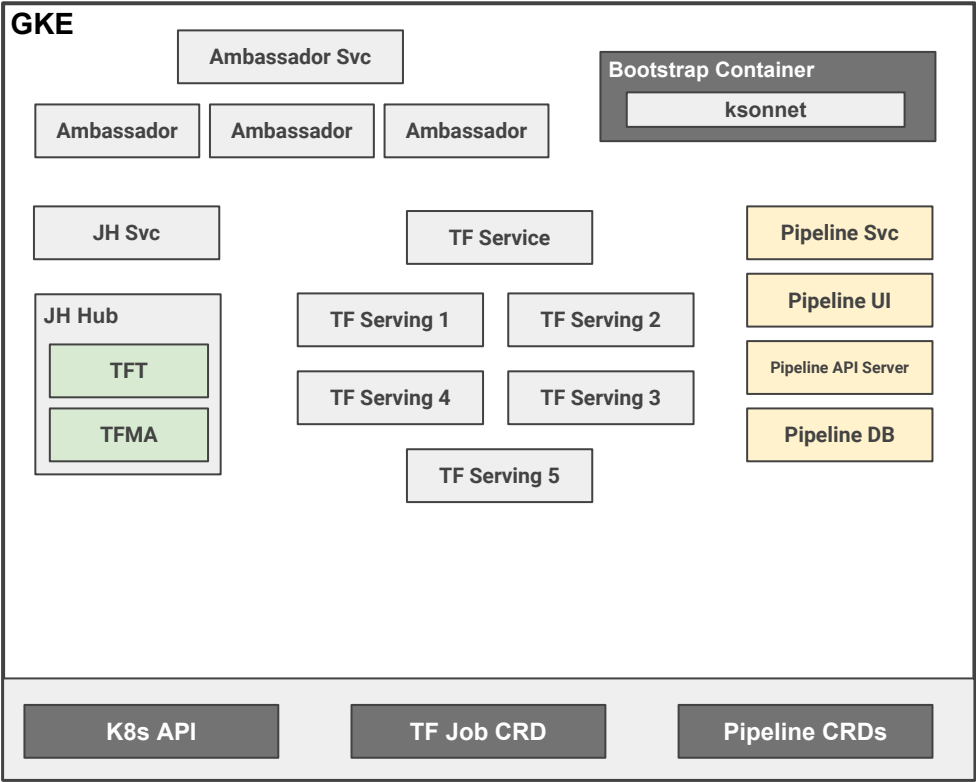
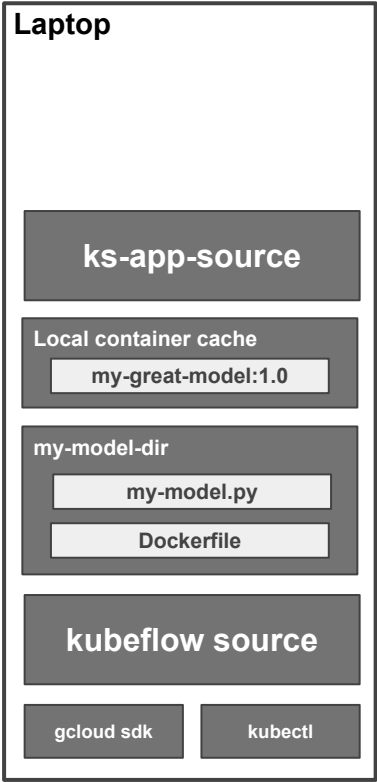
Serving

- First, we need to set a few parameters in for tf-model serving
- These include:
 - `MODEL_COMPONENT=serveMyGreatModelComponent`
 - `MODEL_NAME=my-great-model-name`
 - `MODEL_PATH=gs://PROJECT_ID/my_great_model`
- These are only used during tf-serving creation (not during serving)
- Model path must be publicly accessible

Serving

- Then, we must generate the ksonnet template
 - `be ks generate tf-serving serveMyGreatModelComponent --name=my-great-model-name`
- Set the serving parameters
 - `be ks param set serveMyGreatModelComponent modelPath gs://PROJECT_ID/my_great_model`
- Commit your code to your repo
 - `be 'git add .; git commit -a -m "tf-serving generated"; git push'`
- Deploy the model server
 - `be ks apply kubeflow -c serveMyGreatModelComponent`
- Scale Serving
 - `kubectl scale --replicas=5 deployment/serveMyGreatModel`

Scale Serving



AI Hub



KF Registry



GCR



Block Store



Data Source



GCP Control Plane

```
$ kubectl scale --replicas=5 deployment/serveMyGreatModel
```

Setup a Data Pipeline & Triggering

- First, create a pipeline workflow
 - `vi gcs-bucket-to-tft-pipeline.py`

```
@pipeline
def sentiment_analysis(train_data, eval_data)

    bag_of_words={'review': 'bag_of_words', 'result': 'target'}
    ngrams={ 'review': 'ngrams', 'result': 'target'}
    preprocessor_bow = TFTTransformOp(bag_of_words, train_data, eval_data)
    preprocessor_2gram = TFTTransformOp(ngrams, train_data, eval_data)
    preprocessors = [preprocessor_bow, preprocessor_2gram]

    models, accuracy = [], []
    for p in preprocessors:
        trainer = LinearTrainOp(train_data=p.preprocessed_train, eval_data=p.preprocessed_eval)
        tfma_slicer = TfmaSlicer(trainer.model, p.preprocessed_eval, slicing='review')
        models.add(trainer.model)
        accuracy.add(trainer.accuracy)

    argmin = dsl.ArgMaxOp(
        x=models,
        y=accuracy)

    deployer = DeployerOp(model=argmin.x, endpoint='staging')
```

Setup a Data Pipeline & Triggering

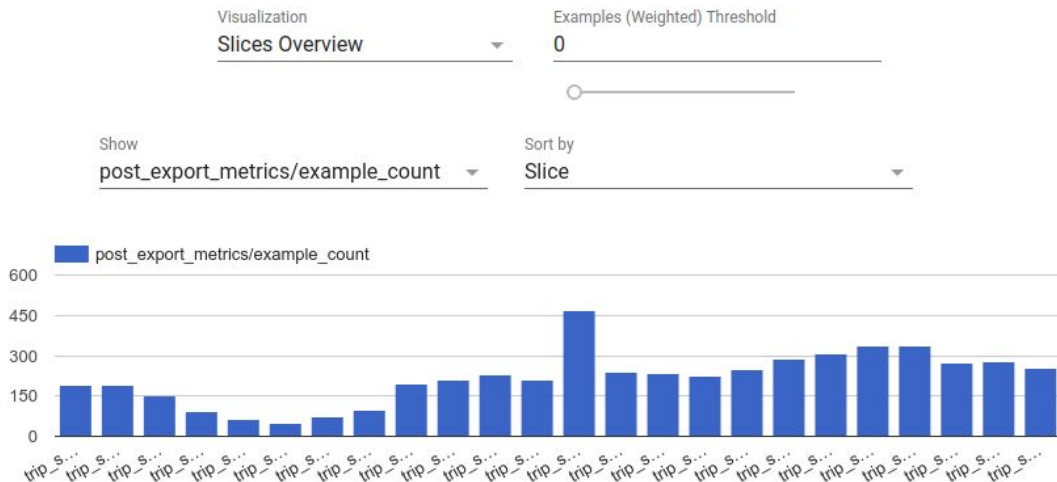
- Next, check your source into your repo
 - `git add .; git commit -a -m "pipeline created"; git push`
- Submit your new pipeline
 - `mlpb submit gcs-bucket-to-tft-pipeline.py`

The next morning...

1. New data gets uploaded to GCS bucket
2. GCS Event Triggers Pipeline
3. Pipelines executes two TFT jobs to download data from GCS bucket
4. GCS Bucket Updated, Workflow Kicks Off
5. TFTs finish and store statistics in GCS
6. TFT processes shut down
7. GCS Event Triggers Training Pipelines
8. Pipelines Trigger TF Jobs
9. TF Jobs write results to store
10. TFT results being written triggers TFMA workflow
11. Pipelines executes TFMA job to download training results and analyze them
12. TFMA writes results to Pipeline DB

We can now read how do the models match up

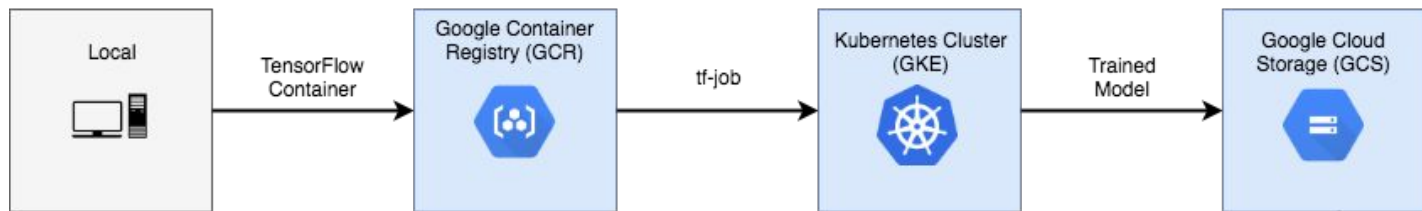
```
In [13]: # Show data sliced along feature column trip_start_hour.
tfma.view.render_slicing_metrics(
    tfma_result_1, slicing_column='trip_start_hour')
```



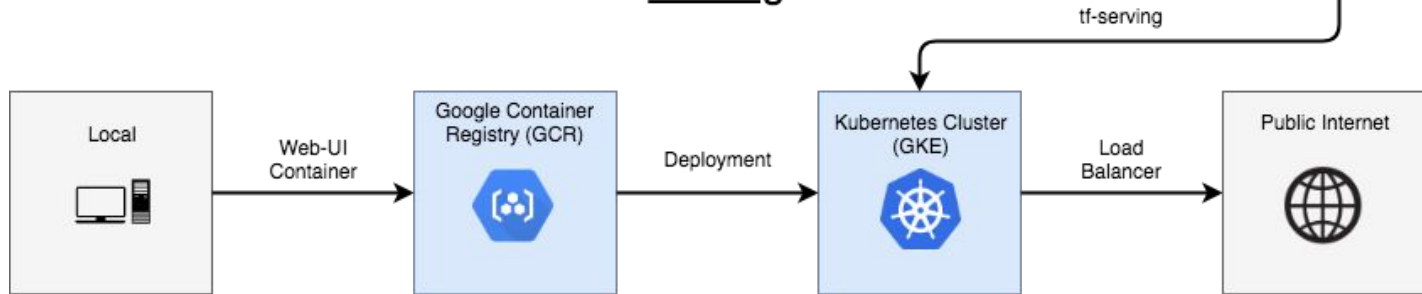
feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:8	0.97938	0.97938	0.66513	0.99010	0.1111
trip_start_hour:9	0.98113	0.98113	0.69231	0.99140	0.0892
trip_start_hour:10	0.95197	0.95197	0.77377	0.98236	0.1541
trip_start_hour:1	0.94180	0.94180	0.78422	0.98231	0.1901

What it does:

Training



Serving



Lab - Introduction to Kubeflow on GKE

Contents

1. **Containers**
2. **Kubernetes**
3. **Google Kubernetes Engine**
4. **Kubeflow - Overview**
5. **Kubeflow - Components**
6. **Kubeflow - Architecture**
7. **ksonnet**
8. **Kubeflow - Tutorial**

Appendix- Tensor2Tensor

ksonnet

Core Concepts: CRDs - Custom Resource Definition

- The CustomResourceDefinition API resource allows you to define custom resources.
- Defining a CRD object creates a new custom resource with a name and schema that you specify.
- The Kubernetes API serves and handles the storage of your custom resource.

Core Concepts: Custom Resource

- A resource is an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind. For example, the built-in pods resource contains a collection of Pod objects.
- A custom resource is an extension of the Kubernetes API that is not necessarily available on every Kubernetes cluster.
- It represents a customization of a particular Kubernetes installation.

Core Concepts: Custom Controller

- Custom resources simply let you store and retrieve structured data.
- Only when combined with a controller that they become a true declarative API.
- A declarative API allows you to declare or specify the desired state of your resource and tries to match the actual state to this desired state. Here, the controller interprets the structured data as a record of the user's desired state, and continually takes action to achieve and maintain this state.
- A custom controller is a controller that users can deploy and update on a running cluster, independently of the cluster's own lifecycle.
- Custom controllers can work with any kind of resource, but they are especially effective when combined with custom resources.

ksonnet

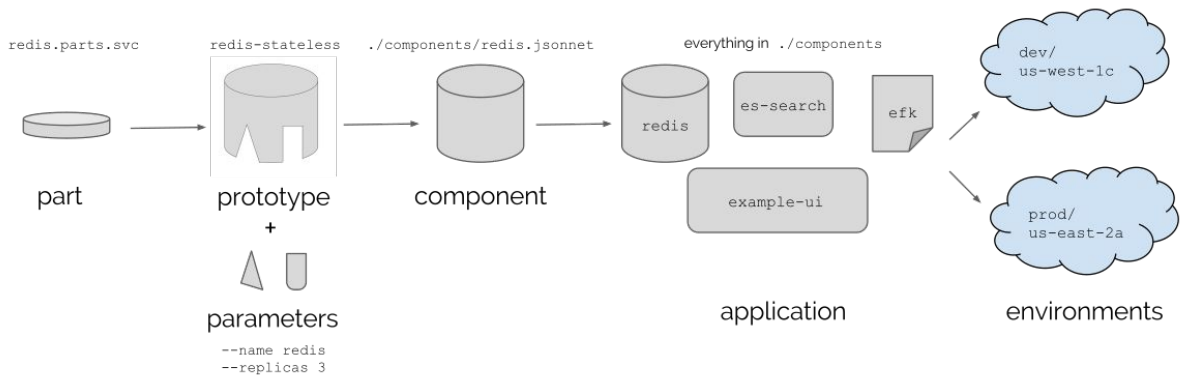


- ksonnet is a framework for writing, sharing, and deploying **Kubernetes application manifests** ([.yaml](#)).
- With its CLI, you can generate a complete application from scratch in only a few commands, or manage a complex system at scale.

Concepts

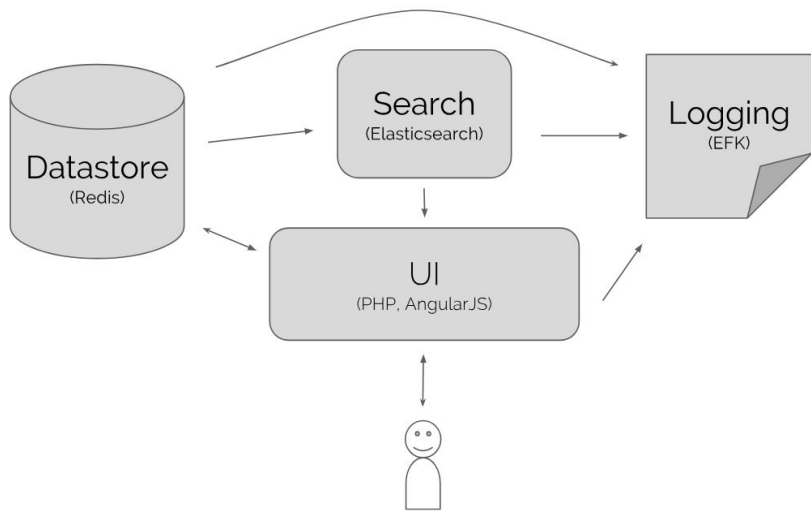
- ksonnet
 - Application
 - Environment
 - Component
 - Prototype
 - Parameter
 - Part
 - Package
 - Registry

- Related
 - Manifest
 - Jsonnet



ksonnet - Application

- An application represents a **well-structured directory of Kubernetes manifests**.
- This directory is autogenerated by **ksonnet init**.
- These manifests typically tie together in some way—for example, they might collectively define a web service like the one shown here



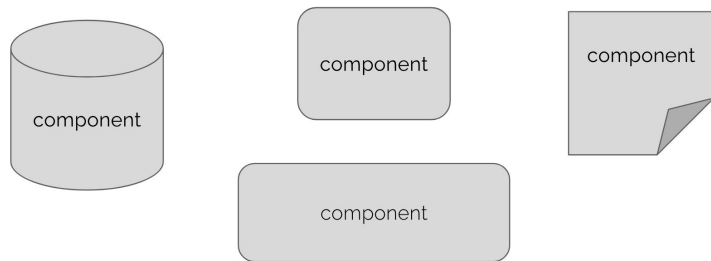
ksonnet - Environment

- An environment consists of four elements, some of which can be pulled from your current kubeconfig context.
- ksonnet allows you to deploy any particular *application* to **multiple** environments

What	Example	Description	How is this tracked?
(1) <i>Name</i>	dev	A string used to identify a particular environment. Must be unique within a ksonnet app.	A directory under <code>environments/</code> (e.g. <code>environments/dev/</code>). A name with slashes results in a hierarchical file structure. For example, <code>us-west/staging</code> creates <code>environments/us-west/staging/</code> .
(2) <i>Server</i>	<code>https://cluster-name-with-hash.us-west-2.elb.amazonaws.com</code>	The address and port of a Kubernetes API server. In other words, identifies a unique cluster.	Inside <code>environments/<env-name>/spec.json</code>
(3) <i>Namespace</i>	dev	Specifically, a Kubernetes namespace .	Inside <code>environments/<env-name>/spec.json</code>
(4) <i>Kubernetes API version</i>	version:v1.7.1	The version of your cluster's Kubernetes API server, defaults to v1.7.0.	Used to generate appropriate files in <code>environments/<env-name>/metadata/</code> based on the specified version of the Kubernetes OpenAPI spec

ksonnet - Component

- An application can be broken down into a series of discrete components
- Components can be as simple as a single Kubernetes resource (e.g. a Deployment) or as complex as a complete logging stack (e.g. EFK).
- More concretely, a component corresponds to a Kubernetes manifest in **components/**

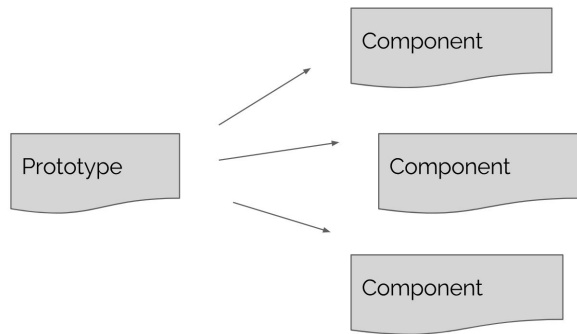


ksonnet - Component

- There are two ways to add this manifest:
 - Typically, you autogenerate it with the **ks generate** command. In this case, the manifest is expressed in a language called Jsonnet.
 - Alternatively, you can manually drop in a file into **components/**. In this case, you can use either JSON or Jsonnet, as long the file is saved with the *.jsonnet extension. (JSON is a subset of Jsonnet so no other changes are needed. YAML is not currently supported, but you can use an open-source CLI tool such as yam2json to convert from YAML to JSON).

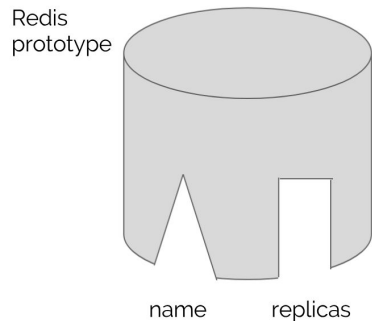
ksonnet - Component - Autogeneration

- How does the autogeneration process work?
 - When you use **ks generate**, the component is generated from a prototype. The distinction between a component and a prototype is a bit subtle. If you are familiar with object oriented programming, you can roughly think of a prototype as a “class”, and a component as its instantiation.
- All of the component files in an app can be deployed to a specified environment using **ks apply**.



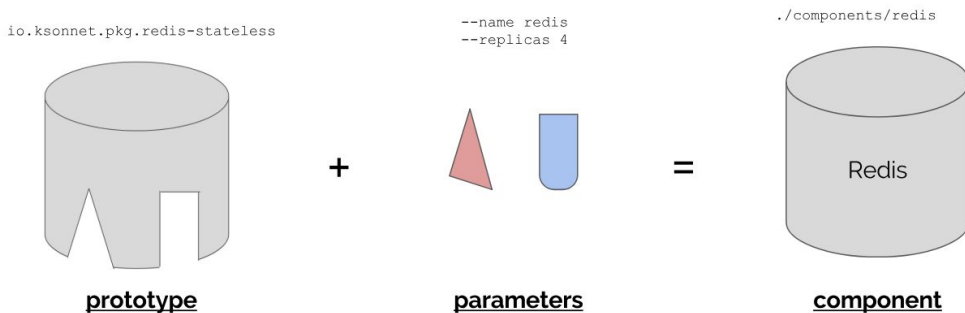
ksonnet - Prototype

- Prototypes are examples that you can use as the basis for your components. By **ks generate**-ing components from prototypes, you can quickly move from a blank app to multiple complete manifests.
- Although you can use these components right away, they are really intended as a starting point. If a prototype-generated component does not fully address your needs, you can directly modify its Jsonnet afterwards (e.g. adding parameters).
- By itself, a prototype is a pre-written but incomplete manifest



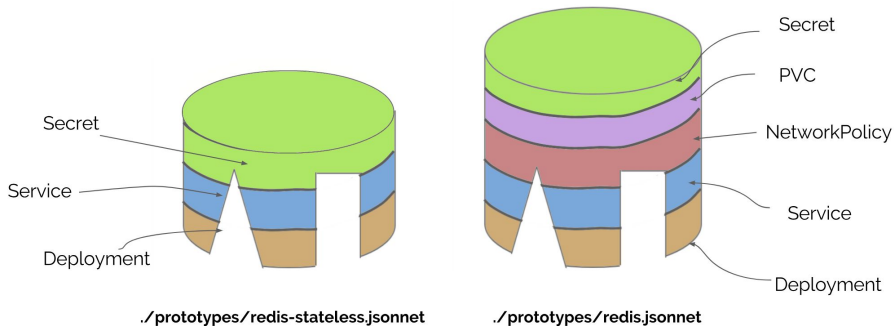
ksonnet - Parameters

- Parameters allow you to customize your prototypes — both at the time of their creation, and after the fact. You can use the various **ks param** commands to view or modify current parameters. Params can be set globally or per-environment.
- Under the hood, the **ks param** commands update a couple of local Jsonnet files, so that you always have a version-controllable representation of what you **ks apply** onto your Kubernetes cluster.

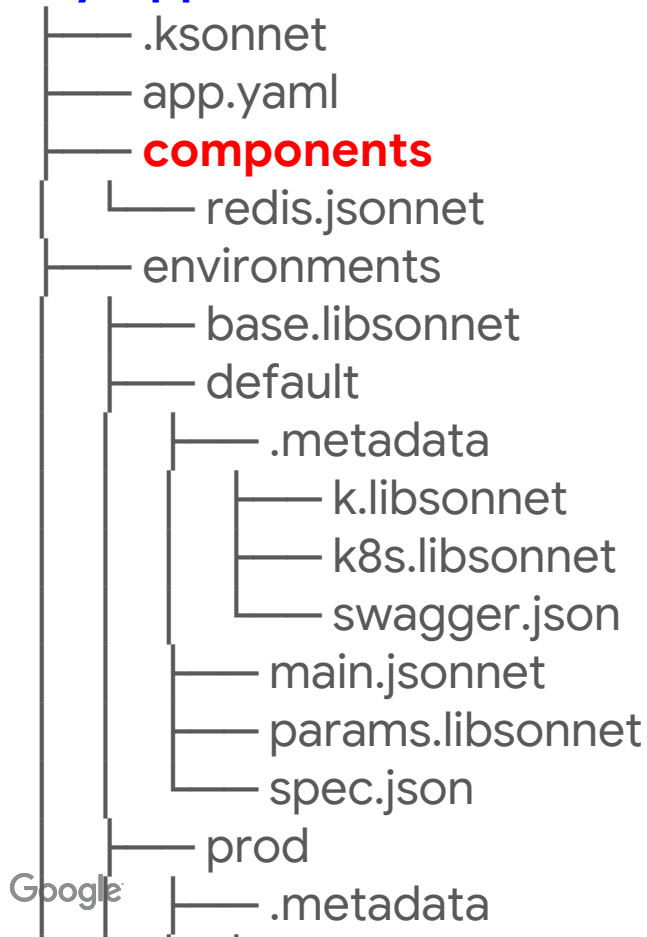


ksonnet - Part

- Prototypes often come in sets, where each prototype offers a slightly different “flavor” of a common base (e.g. redis-persistent and redis-stateless are two possible ways of deploying a Redis datastore).
- Given that ksonnet is designed around modularity, a common pattern is to refactor most of the shared prototype code into a single parts library
- Once a library like this is established, different parts like `redis.parts.deployment.persistent` can be mixed and matched to produce new prototypes



my-app

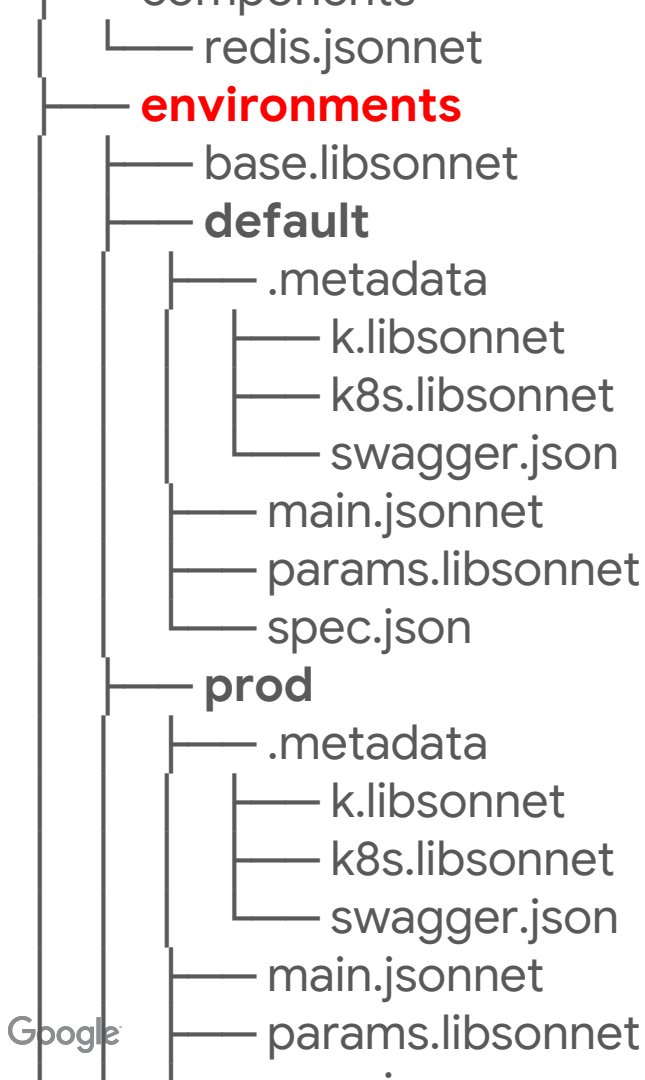


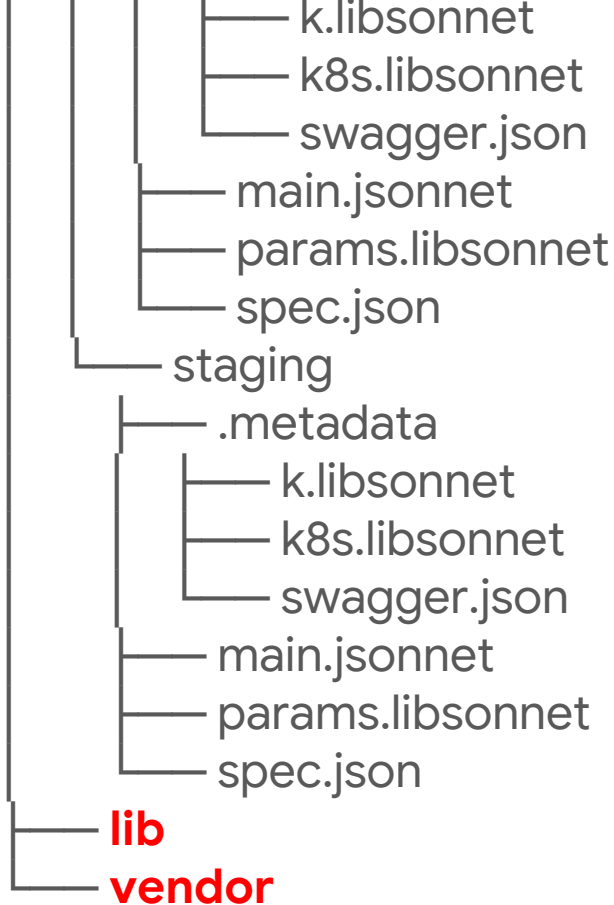
ksonnet - Folder Structure

- **components** have a flat hierarchy, residing in the components directory.
- **components** are shared across all environments
- a **component** corresponds to a Kubernetes manifest

ksonnet - Folder Structure

- **environments** are "registered" if there exists a subdirectory in environments with the spec.json file. spec.json contains environment details, such as the cluster server URI and cluster namespace.
- **environments** each contain their own .metadata folder containing details of the ksonnet-lib and kubernetes versions being used by the respective environment.



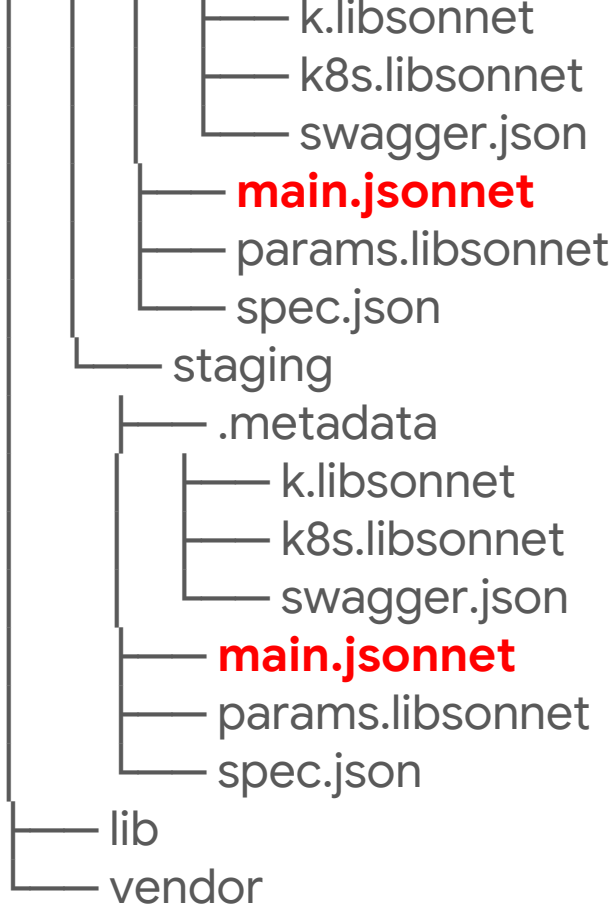


ksonnet - Folder Structure

- The **lib/** folder contains user-written .libsonnet files
- The **vendor/** folder contains libraries that define prototypes and their constituent parts

ksonnet - Jsonnet

- ksonnet uses **.jsonnet** files to express all of your components/manifests (and associated configuration files, like components/params.libsonnet).



ksonnet - Jsonnet

- Jsonnet is a data templating language — for simplicity, think of it as a fancier superset of JSON, one that supports features like variables and object concatenation. If you've ever had to copy and paste large chunks of YAML or JSON manifests, Jsonnet avoids this problem!

Jsonnet

 Copy

```
local nginxContainer =  
  container.new("nginx", "nginx:1.13.0") +  
  container.ports(containerPort.newNamed("http", 80));  
  
deployment.new("nginx", 5, nginxContainer, {app: "nginx"})
```

YAML

 Copy

```
apiVersion: apps/v1beta1  
kind: Deployment  
metadata:  
  name: nginx  
spec:  
  replicas: 5  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
        - image: 'nginx:1.13.0'  
          name: nginx  
          ports:  
            - containerPort: 80  
              name: http
```

Lab - Kubeflow End to End

Next steps?

- Use the remaining credits to upskill in qwiklabs
- Check out kubeflow.org
- Star <https://github.com/kubeflow/kubeflow>
- Join slack: kubeflow (<http://kubeflow.slack.com>)
- Follow twitter: @kubeflow
- Email/Connect
 - feixue@google.com
 - chrischo@google.com

Appendix - Tensor2Tensor T2T

What is Tensor2Tensor?

- Batteries-included toolkit for machine learning research and practice
- Is a library of deep learning models and datasets designed to make deep learning more accessible and accelerate ML research
- T2T is actively used and maintained by researchers and engineers within the Google Brain team and a community of users

What is Tensor2Tensor?

- It comes equipped with key ingredients including hyperparameters, data-sets, model architectures and learning rate decay schemes
- Any of these components can be swapped in and out in a **modular fashion** without completely destroying everything. From a training perspective, this means that with T2T you can bring in new models and datasets at any time

What is Tensor2Tensor?

- Library of:
 - Models (45 and counting)
 - Hyperparameter sets
 - Datasets (193 and counting, standardized on TFRecord files)
 - Model & training components
- Scripts to:
 - Generate datasets: `t2t-datagen`
 - Train and evaluate: `t2t-trainer`
 - Decode: `t2t-decoder`

Components

Datasets

- **Datasets** are all standardized on **TFRecord** files with tensorflow.
- All datasets are registered and generated with the **data generator** and many common sequence datasets are already available for generation and use

Problems and Modalities

- **Problems** define training-time hyperparameters for the dataset and task, mainly by setting input and output **modalities** (e.g. symbol, image, audio, label) and vocabularies, if applicable. All problems are defined either in `problem_hparams.py` or are registered with `@registry.register_problem` (run `t2t-datagen` to see the list of all available problems).
- **Modalities**, defined in `modality.py`, abstract away the input and output data types so that **models** may deal with modality-independent tensors.

Models

- **T2TModel** 's define the core tensor-to-tensor transformation, independent of input/output modality or task.
- Models take dense tensors in and produce dense tensors that may then be transformed in a final step by a **modality** depending on the task (e.g. fed through a final linear transform to produce logits for a softmax over classes).
- All models are imported in the models subpackage, inherit from **T2TModel** - defined in `t2t_model.py` - and are registered with `@registry.register_model`.

Hyperparameter Sets

- **Hyperparameter sets** are defined and registered in code with `@registry.register_hparams` and are encoded in `tf.contrib.training.HParams` objects. The `HParams` are available to both the problem specification and the model.
- A basic set of hyperparameters are defined in `common_hparams.py` and hyperparameter set functions can compose other hyperparameter set functions.

Trainer

- The **trainer** binary is the main entrypoint for training, evaluation, and inference. Users can easily switch between problems, models, and hyperparameter sets by using the **--model**, **--problem**, and **--hparams_set** flags.
- Specific hyperparameters can be overridden with the **--hparams** flag. **--schedule** and related flags control local and distributed training/evaluation (distributed training documentation).

Adding your own components

- T2T's components are registered using a central registration mechanism that enables easily adding new ones and easily swapping amongst them by command-line flag.
- You can add your own components without editing the T2T codebase by specifying the `--t2t_usr_dir` flag in `t2t-trainer`.
- You can do so for models, hyperparameter sets, modalities, and problems. Please do submit a pull request if your component might be useful to others.

Training and Evaluating in T2T

Local - Command Line

Installing T2T

```
pip install tensor2tensor
```

```
# See what problems, models, and hyperparameter sets are available.
```

```
# You can easily swap between them (and add new ones).
```

```
t2t-trainer --registry_help
```

Setup variables, directories

```
PROBLEM=translate_ende_wmt32k
```

```
MODEL=transformer
```

```
HPARAMS=transformer_base_single_gpu
```

```
DATA_DIR=$HOME/t2t_data
```

```
TMP_DIR=/tmp/t2t_datagen
```

```
TRAIN_DIR=$HOME/t2t_train/$PROBLEM/$MODEL-$HPARAMS
```

```
mkdir -p $DATA_DIR $TMP_DIR $TRAIN_DIR
```

Generate Data

```
t2t-datagen \  
  --data_dir=$DATA_DIR \  
  --tmp_dir=$TMP_DIR \  
  --problem=$PROBLEM
```

Train

```
t2t-trainer \  
  --data_dir=$DATA_DIR \  
  --problem=$PROBLEM \  
  --model=$MODEL \  
  --hparams_set=$HPARAMS \  
  --output_dir=$TRAIN_DIR
```

Create some quick data for testing

```
# Creating Testing Data
```

```
DECODE_FILE=$DATA_DIR/decode_this.txt
```

```
echo "Hello world" >> $DECODE_FILE
```

```
echo "Goodbye world" >> $DECODE_FILE
```

```
# Creating Verification Data
```

```
echo -e 'Hallo Welt\nAuf Wiedersehen Welt' > ref-translation.de
```

```
# Size of beam
```

```
BEAM_SIZE=4
```

```
# Alpha for length penalty
```

```
ALPHA=0.6
```

Inference and Decoding

```
# Encoding - Input str to features dict, ready for inference
```

```
# Decoding - List of ints to str
```

```
t2t-decoder \  
  --data_dir=$DATA_DIR \  
  --problem=$PROBLEM \  
  --model=$MODEL \  
  --hparams_set=$HPARAMS \  
  --output_dir=$TRAIN_DIR \  
  --decode_hparams="beam size=$BEAM_SIZE,alpha=$ALPHA" \  
  --decode_from_file=$DECODE_FILE \  
  --decode_to_file=translation.en
```

Evaluate the BLEU score for inference

```
# Note: Report this BLEU score in papers, not the internal approx_bleu metric.  
t2t-bleu --translation=translation.en --reference=ref-translation.de
```

Training and Evaluating in T2T

Local - Python T2T Library

Colab demo

<https://colab.research.google.com>

Training and Evaluating in T2T

Google Cloud ML Engine

Train and evaluate

```
t2t-trainer \  
  --data_dir=$DATA_DIR \  
  --problems=$PROBLEM \  
  --model=$MODEL \  
  --hparams_set=$HPARAMS \  
  --output_dir=$TRAIN_DIR
```

Train and evaluate on Cloud ML Engine

```
t2t-trainer \  
  --data_dir=$DATA_DIR \  
  --problems=$PROBLEM \  
  --model=$MODEL \  
  --hparams_set=$HPARAMS \  
  --output_dir=$TRAIN_DIR \  
  --cloud_mlengine \  
  --worker_gpu=4
```

Hyperparameter Tune on Cloud ML Engine

```
t2t-trainer \  
  --data_dir=$DATA_DIR \  
  --problems=$PROBLEM \  
  --model=$MODEL \  
  --hparams_set=$HPARAMS \  
  --output_dir=$TRAIN_DIR \  
  --cloud_mlengine \  
  --worker_gpu=4 \  
  --hparams_range=transformer_base_range \  
  --autotune_parallel_trials=3 \  
  --autotune_objective=$METRIC_NAME \  
  --autotune_maximize \  
  --autotune_max_trials=100
```

Train and evaluate on Cloud TPU

```
t2t-trainer \  
  --data_dir=$DATA_DIR \  
  --problems=$PROBLEM \  
  --model=$MODEL \  
  --hparams_set=$HPARAMS \  
  --output_dir=$TRAIN_DIR \  
  --cloud_tpu
```

It's a collaboration!

**Ashish Vaswani¹, Samy Bengio¹, Eugene Brevdo¹, Francois Chollet¹, Aidan N. Gomez¹,
Stephan Gouws¹, Llion Jones¹, Łukasz Kaiser^{1, 3}, Nal Kalchbrenner², Niki Parmar¹,
Ryan Sepassi^{1, 4}, Noam Shazeer¹, and Jakob Uszkoreit¹**

¹Google Brain

²DeepMind

³Corresponding author: lukaszkaizer@google.com

⁴Corresponding author: rsepassi@google.com

We wrote a paper!

<https://arxiv.org/abs/1803.07416>

We're on GitHub!

<https://github.com/tensorflow/tensor2tensor>