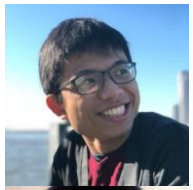
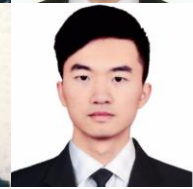
Serverless Compute Platforms on Kubernetes:

Beyond Web Applications

Alex Glikson

Senior Research Architect, Cloud Platforms
Carnegie Mellon University
(IBM Research, Israel)



In collaboration with Ping-Min Lin, Shengjie Luo, Ke Chang, Shichao Nie

KubeCon, May 2019

Outline

- Introduction
  - Serverless
    - Serverless Compute
      - FaaS
      - Non-FaaS
- Our Use-Cases
  - Deep Learning
  - Interactive Computing
    - Demo
- Conclusions

Serverless

- Many definitions
- In a nutshell:
- **Avoid** management of **servers**, as a representative example of tasks that:
  - Keep you **distracted** from developing your *core* business capabilities, and
  - Can be **outsourced** to someone you trust, for whom this *would* be their core business
- Serverless = Distruction-Free
  - Separation of concerns

Serverless = Distruction-Free (Examples)

- Object Storage:
  - Core: storage of unstructured data objects
  - Distruction: servers, storage, network, high availability, fault tolerance, replication, consistency

Example:
Amazon S3

- Micro-services:
  - Core: services logic, interfaces
  - Distruction: infra, scaling, LB, HA/FT, API management, routing, service discovery, etc

Example:
Kubernetes+Istio

- Async/Event-driven:
  - Core: events, processing logic
  - Distruction: eventing, messaging, queuing, notifications, etc (+infra/scaling/LB/HA/FT/auth/etc)

Example:
Lambda, SNS, etc

- …

Serverless Compute Platform (SCP)

- Platform that executes user-provided **code** (BYOC)
- Distruction-free
  - Simplified **management**
    - Deployment, scaling, metering, monitoring, logging, updates, etc
  - Seamless **integration** with services that the ‘compute’ interacts with (or depends on)
    - Event sources, data sources, communication middleware, etc.
- Often optimized for specific **application patterns**
- **Elasticity / Pay-per-use**

SCP: Function as a Service (FaaS)

Platform Property	General-Purpose FaaS
Examples	- Lambda, Azure functions, Google Functions; - Kubeless, OpenFaaS, OpenWhisk
Code	Arbitrary functions (packaging and runtime constraints slightly differ among providers)
Management	Fully managed isolated runtime containers
Integration	Seamless integration with multiple event sources
Application Pattern	Short-lived, ephemeral functions; High load variability; Often low sensitivity to latency
Elasticity, Pay-per-use	Per-request scaling and metering (e.g., 100ms granularity in Lambda)

SCP: Specialized (Embedded) FaaS

Platform Property	Programmable Event-driven platforms	Programmable Network edge platforms	Programmable Workflow engines
Examples	Trillio Functions, Github Actions	PubNub Functions, Lambda@Edge	NodeRED, IBM Watson Streaming Pipelines
Code			
Management			
Integration			
Application Pattern			
Elasticity, Pay-per-use			

SCP: Non-FaaS Platforms

Platform Property	Serverless ETL	Cloud-Native Web Applications
Examples	AWS Glue	Knative
Code	PySpark, PyShell jobs	Arbitrary code + Dockerfile
Management	“Serverless” Spark cluster; Python runtime	K8s features + code-to-deploy
Integration	Data catalogue	Service mash, build, eventing
Application Pattern	Periodic or ad-hoc Spark jobs Non-parallel pre/post-processing jobs	Long-running, scale-out services Linear resource demand per request
Elasticity, Pay-per-use	Per-job scaling and metering	Request-based scaling, incl. to zero

Outline

- Introduction
  - Serverless
    - Serverless Compute
      - FaaS
      - Non-FaaS
- Our Use-Cases
  - Deep Learning
  - Interactive Computing
    - Demo
- Conclusions

Deep Learning

transportation

medicine

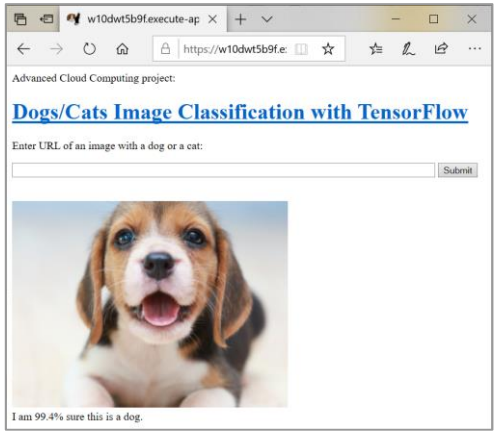
smart cities, security

consumer

games

e-commerce

- Resource-intensive
  - (1) model training, (2) inference
- Frameworks: Tensorflow, Keras, PyTorch, etc.
- Example application: Image Classification
  - Given a model + unlabeled example(s), predict label(s)
  - Compute-intensive, scale-out, can leverage GPUs
  - Accessed via HTTP-based API (returns HTML or JSON)



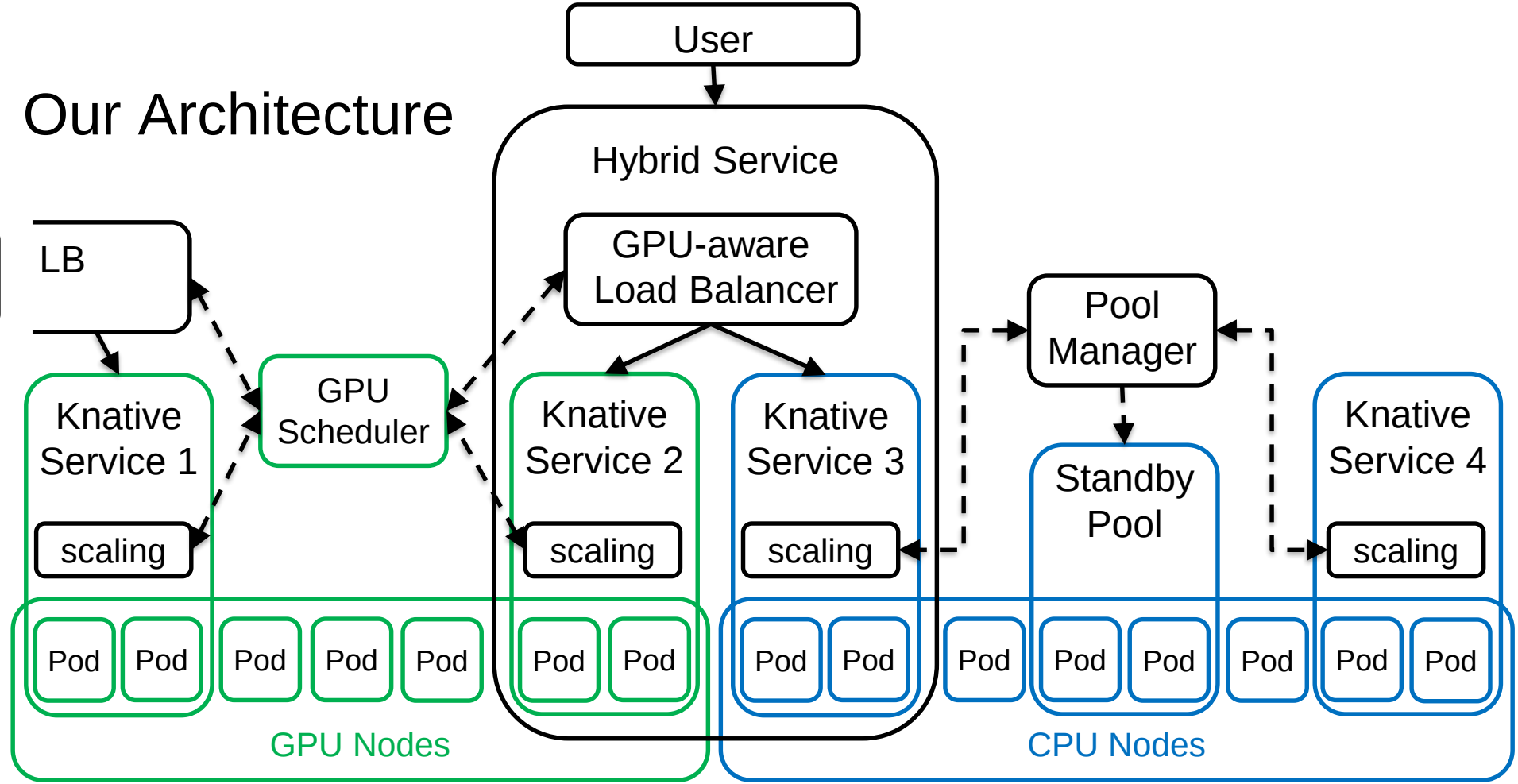
Kubeflow: Kubernetes-based platform for ML/DL

- Fully managed training jobs (via CRDs)
- Deployment of tfserving
- Jupyter Notebooks

SCP for Deep Learning Inference

Platform Property	Kubeflow	Our solution (leveraging Knative)
Code	No code, using tf-serving	Model inference implementation (Python)
Management	Life cycle of the model, provisioning of k8s Deployment serving the model	Enhanced Knative build (code-to-deploy) Knative serving (versioning, rollout, etc)
Integration	K8s ecosystem; model storage	K8s, istio
Application Pattern	Scale-out cluster of long-running tf-serving containers	Optimization: - mitigate cold starts by pooling warm containers - load-balance between CPU and GPU nodes
Elasticity, Pay-per-use	Regular K8s Deployment (e.g., HPA)	Request-based scaling with Knative (+optimization), including scaling to zero

Our Architecture



Design Details

- Build: Automatically add HTTP interface
  - Augment the provided inference logic with a Django ‘wrapper’, then use Knative build to deploy it
- Load-balancing across GPU-enabled and CPU-only nodes
  - Patch Knative to support GPU resources
  - Based on model properties, indicate in the Knative service template whether a GPU is preferable
  - Two-level scheduling: 1 GPU service and 1 CPU service for each app; fair time-sharing of GPUs
- Maintain a pool of ‘warm’ Pods
  - “Pool” is a ReplicaSet with ‘warm’ (running) Pods
    - Size is adjusted dynamically by the Pool Controller (cluster utilization, estimated demand)
  - Knative scaling logic consumes a warm Pod from the Pool instead of provisioning a new one
    - Pod “migration” is implemented by label manipulation + update of the Istio side-car via API

Lessons Learned

- Standardized HTTP wrappers can be used to deliver FaaS-like experience
  - Can leverage existing open source FaaS solutions (e.g., OpenWhisk)
- More fine-grained management of GPU resources would be beneficial
  - The overhead of 2-level scheduling is substantial
- For reuse of ‘warm’ Pods, stronger notion of ‘similarity’ between Pods is needed
  - E.g., same model version?
- Even pool of size 1 significantly reduces the chances of cold starts
  - Instead of pools, can we reuse priority classes and make Knative scaling logic adjust priorities?

Outline

- Introduction
  - Serverless
    - Serverless Compute
      - FaaS
      - Non-FaaS
- Our Use-Cases
  - Deep Learning
  - Interactive Computing
    - Demo
- Conclusions

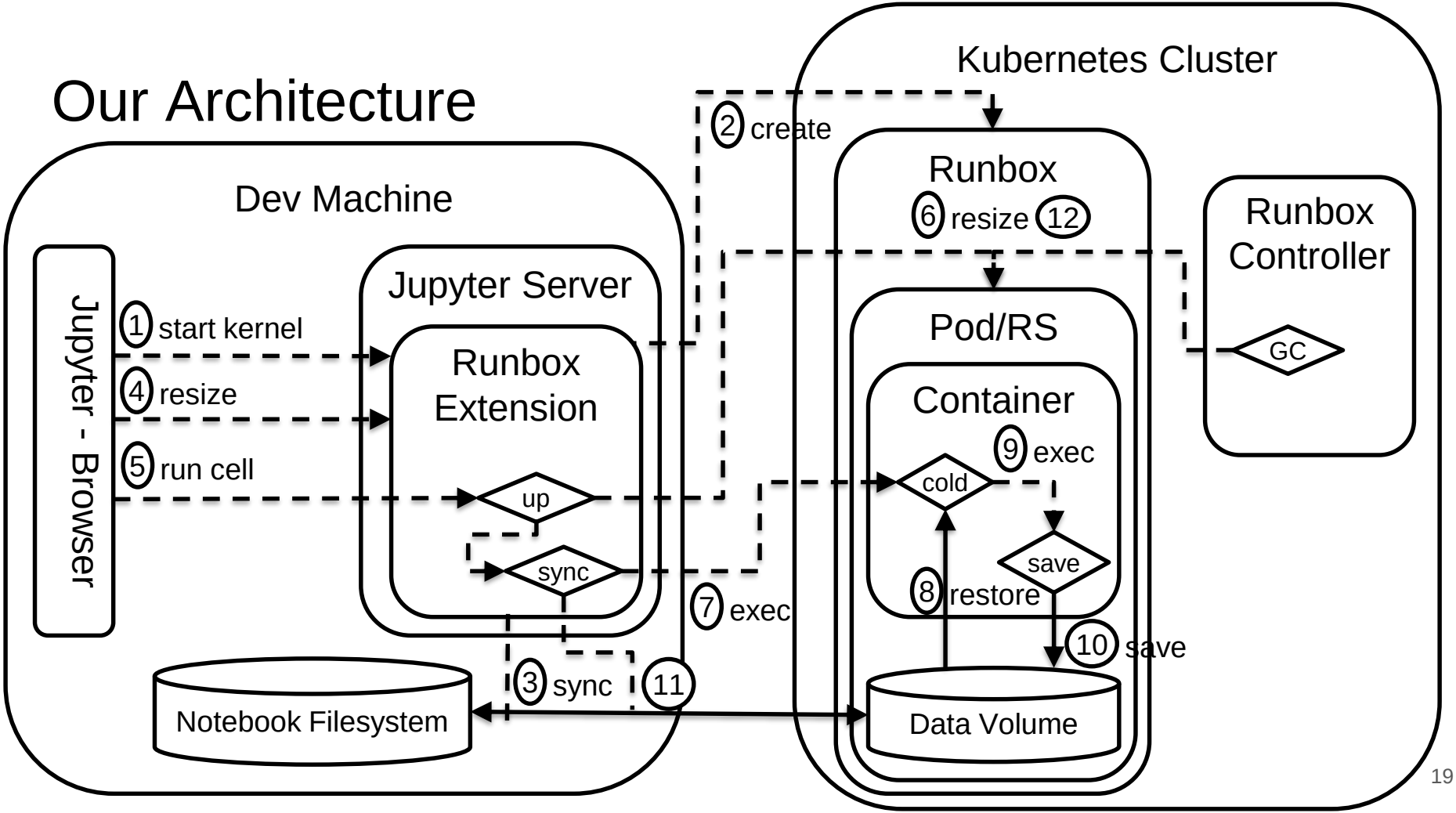
Interactive Computing

- Iterative invocation of computation-intensive stateful ad-hoc tasks, triggered by explicit user interaction
  - Often not designed for scale-out
- Example: Data Science using Jupyter Notebook
  - Other examples: Linux Shell (arbitrary shell interactions), IDE (build/execution of code)

SCP for Interactive Computing

Platform Property	Kubeflow Jupyter Notebooks	Our Solution (with Jupyter, Kubernetes)
Code	IPython, Bash, etc	Python, Bash (with some restrictions)
Management	Provisioning, life cycle of Jupyter servers and Notebooks	Provisioning and management of resources used by Notebooks
Integration	Data sources	Data sources
Application Pattern	Interactive data science (fixed resources)	Interactive data science; flexible resource allocation guided by user input (e.g., magics)
Elasticity, Pay-per-use	No scaling (fixed resource allocation per Notebook)	Vertical scaling, including scaling to zero, guided by user input (+optimization)

Our Architecture



Design Details

- Special Jupyter Kernels, delegating execution to a K8s Pod using `kubectl exec`
  - E.g., scp-python, scp-bash
- State is persisted in a K8s volume attached to the Pod
  - Snapshot/restore in-memory state using `dill` in Python and `set/source` in Bash
  - Also, state is synchronized from/to the local machine via a side-car running unison
- Pod is scaled down (optionally, to zero) when nothing is executed
  - E.g., by scaling the containing ReplicaSet, or using in-place Pod vertical scaling (WIP)
  - Tradeoff between capacity for ‘warm’ containers and latency managed by dedicated controller
- When image changes (e.g., after `apt install`), a new image is committed
  - Using tags for versioning; docker-squash to remove redundant layers
- Magics to control the non-functional properties
  - E.g., resource allocation, whether or not image snapshot is needed, etc

Demo

Lessons Learned

- Kubernetes originally focused on scale-out workloads, but can also support scale-up
    - New kind of controller?
- Generic support for application-assisted snapshots could be useful

- For use-cases involving ephemeral compute, API for direct access to volumes could be useful

Outline

- Introduction
  - Serverless
    - Serverless Compute
      - FaaS
      - Non-FaaS
- Our Use-Cases
  - Deep Learning
  - Interactive Computing
    - Demo
- Conclusions

Conclusions

- “Serverless” = BYOC + elasticity + distruction-free
- “Serverless” derives different requirements for different workloads
- Knative can be enhanced to achieving “serverless” goals for DL inference
  - Aligned with KFserving goals?
- SCP for Interactive Computing requires new capabilities on top of Kubernetes
- Lots of opportunities to deliver ‘serverless’ experience for new workloads!