Serverless Compute Platforms on Kubernetes:

Beyond Web Applications

Alex Glikson

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

KubeCon, May 2019

with Ping-Min Lin (Pinterest), Shengjie Luo (VMware), Ke Chang (Facebook), Shichao Nie (Alibaba)

Outline

- Introduction
  - Serverless
    - Serverless Compute
      - FaaS
      - Non-FaaS
- Our Use-Cases
  - Interactive Computing
  - Deep Learning
- 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 = Distraction-Free
  - Separation of concerns

Serverless = Distraction-Free (Examples)

- Object Storage:
  - Core: data organization
  - Distraction: servers, storage, network, high availability, fault tolerance, replication, consistency

Example:
Amazon S3

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

Example:
Kubernetes+Istio

- Async/Event-driven:
  - Core: event-processing logic
  - Distraction: 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)

- Often optimized for specific **application patterns**

- Distraction-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.

- Bonus: **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	
Application Pattern	
Management	
Integration	
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)
Application Pattern	
Management	
Integration	
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)
Application Pattern	Short-lived, ephemeral functions, triggered by events or requests; Often: high load variability, low sensitivity to latency
Management	
Integration	
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)
Application Pattern	Short-lived, ephemeral functions, triggered by events or requests; Often: high load variability, low sensitivity to latency
Management	Fully managed isolated runtime containers (provisioning, monitoring, logging, etc)
Integration	
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)
Application Pattern	Short-lived, ephemeral functions, triggered by events or requests; Often: high load variability, low sensitivity to latency
Management	Fully managed isolated runtime containers (provisioning, monitoring, logging, etc)
Integration	Seamless integration with multiple event sources
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)
Application Pattern	Short-lived, ephemeral functions, triggered by events or requests; Often: high load variability, low sensitivity to latency
Management	Fully managed isolated runtime containers (provisioning, monitoring, logging, etc)
Integration	Seamless integration with multiple event sources
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	…
Examples	Trillio Functions, Github Actions	PubNub Functions, Lambda@Edge	…
Code	Arbitrary functions (programming languages often limited)		
Application Pattern	Short-lived, ephemeral functions, triggered by events or requests;		
Management	Fully managed isolated runtime		
Integration	The hosting platform		
Elasticity, Pay-per-use	Per-request scaling and metering		

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	Serverless ETL
Examples	
Code	
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	Serverless ETL
Examples	AWS Glue
Code	
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	Serverless ETL
Examples	AWS Glue
Code	PySpark, PyShell jobs
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	Serverless ETL
Examples	AWS Glue
Code	PySpark, PyShell jobs
Application Pattern	Data-parallel Spark jobs (periodic or ad-hoc) Non-parallel pre/post-processing jobs
Management	
Integration	
Elasticity, Pay-per-use	

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	**Serverless ETL**
Examples	AWS Glue
Code	PySpark, PyShell jobs
Application Pattern	Data-parallel Spark jobs (periodic or ad-hoc) Non-parallel pre/post-processing jobs
Management	Fully managed Spark cluster; Python runtime
Integration	
Elasticity, Pay-per-use	

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	Serverless ETL
Examples	AWS Glue
Code	PySpark, PyShell jobs
Application Pattern	Data-parallel Spark jobs (periodic or ad-hoc) Non-parallel pre/post-processing jobs
Management	Fully managed Spark cluster; Python runtime
Integration	Data catalogue
Elasticity, Pay-per-use	

Other (Non-FaaS?) SCPs: Serverless ETL

Platform Property	Serverless ETL
Examples	AWS Glue
Code	PySpark, PyShell jobs
Application Pattern	Data-parallel Spark jobs (periodic or ad-hoc) Non-parallel pre/post-processing jobs
Management	Fully managed Spark cluster; Python runtime
Integration	Data catalogue
Elasticity, Pay-per-use	Per-job scaling and metering

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	**Cloud-Native Web Applications**
Examples	
Code	
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	Cloud-Native Web Applications
Examples	Knative
Code	
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	Cloud-Native Web Applications
Examples	Knative
Code	Arbitrary application serving HTTP requests
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	Cloud-Native Web Applications
Examples	Knative
Code	Arbitrary application serving HTTP requests
Application Pattern	Long-running, scale-out services; Linear resource demand per request Often high-throughput, low-latency
Management	
Integration	
Elasticity, Pay-per-use	

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	Cloud-Native Web Applications
Examples	Knative
Code	Arbitrary application serving HTTP requests
Application Pattern	Long-running, scale-out services; Linear resource demand per request Often high-throughput, low-latency
Management	K8s features + code-to-deploy, revisions, canary deployment, etc
Integration	
Elasticity, Pay-per-use	

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	Cloud-Native Web Applications
Examples	Knative
Code	Arbitrary application serving HTTP requests
Application Pattern	Long-running, scale-out services; Linear resource demand per request Often high-throughput, low-latency
Management	K8s features + code-to-deploy, revisions, canary deployment, etc
Integration	Service mash, build, eventing
Elasticity, Pay-per-use	

Non-FaaS SCP: Cloud-Native Web Applications

Platform Property	Cloud-Native Web Applications
Examples	Knative
Code	Arbitrary application serving HTTP requests
Application Pattern	Long-running, scale-out services; Linear resource demand per request Often high-throughput, low-latency
Management	K8s features + code-to-deploy, revisions, canary deployment, etc
Integration	Service mash, build, eventing
Elasticity, Pay-per-use	Request-based scaling, incl. to zero

What Other Application Patterns Could Justify a Specialized SCP?

Platform Property	**?**
Examples	?
Code	?
Application Pattern	?
Management	?
Integration	?
Elasticity, Pay-per-use	?

Outline

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

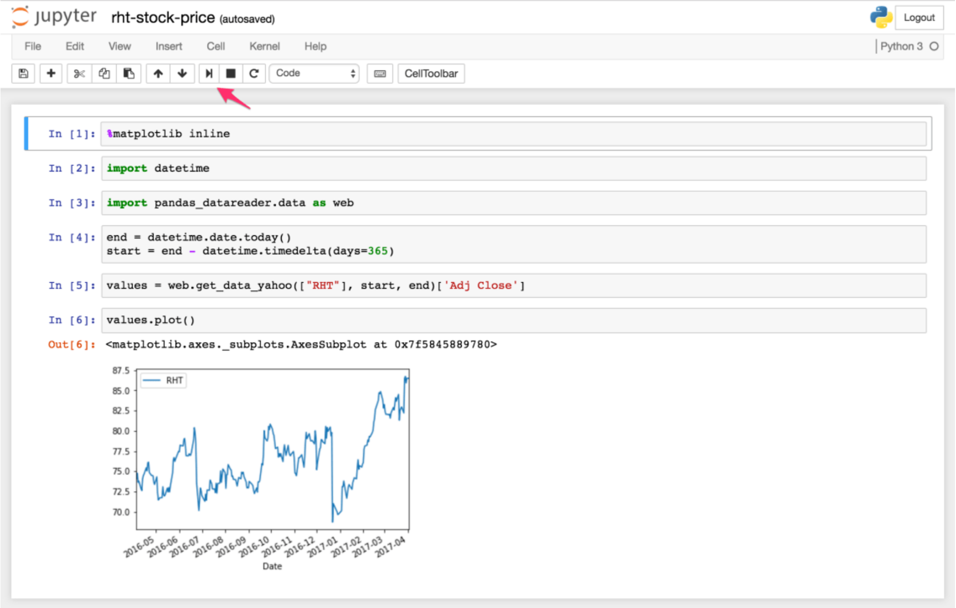
Interactive Computing

- Example: Data Science using Jupyter Notebook

- Architecture 1: Python + Spark
  - Scale-out Spark jobs
  - Requires Spark programming model

- Architecture 2: “pure” Python
  - Local execution, using non-parallel Python libraries
  - Not designed for scale-out, but can take advantage of scale-up

- Other example: Linux Shell



SCP for Interactive Computing

Property	Interactive Computing (Jupyter, Shell)
Code	
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

SCP for Interactive Computing

Property	Interactive Computing (Jupyter, Shell)
Code	Python, Bash
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

SCP for Interactive Computing

Property	Interactive Computing (Jupyter, Shell)
Code	Python, Bash
Application Pattern	Iterative invocation of stateful, non-parallel, computation-intensive, ad-hoc tasks, triggered by explicit user interaction
Management	
Integration	
Elasticity, Pay-per-use	

SCP for Interactive Computing

Property	Interactive Computing (Jupyter, Shell)
Code	Python, Bash
Application Pattern	Iterative invocation of stateful, non-parallel, computation-intensive, ad-hoc tasks, triggered by explicit user interaction
Management	Provisioning, management, scaling of underlying resources
Integration	
Elasticity, Pay-per-use	

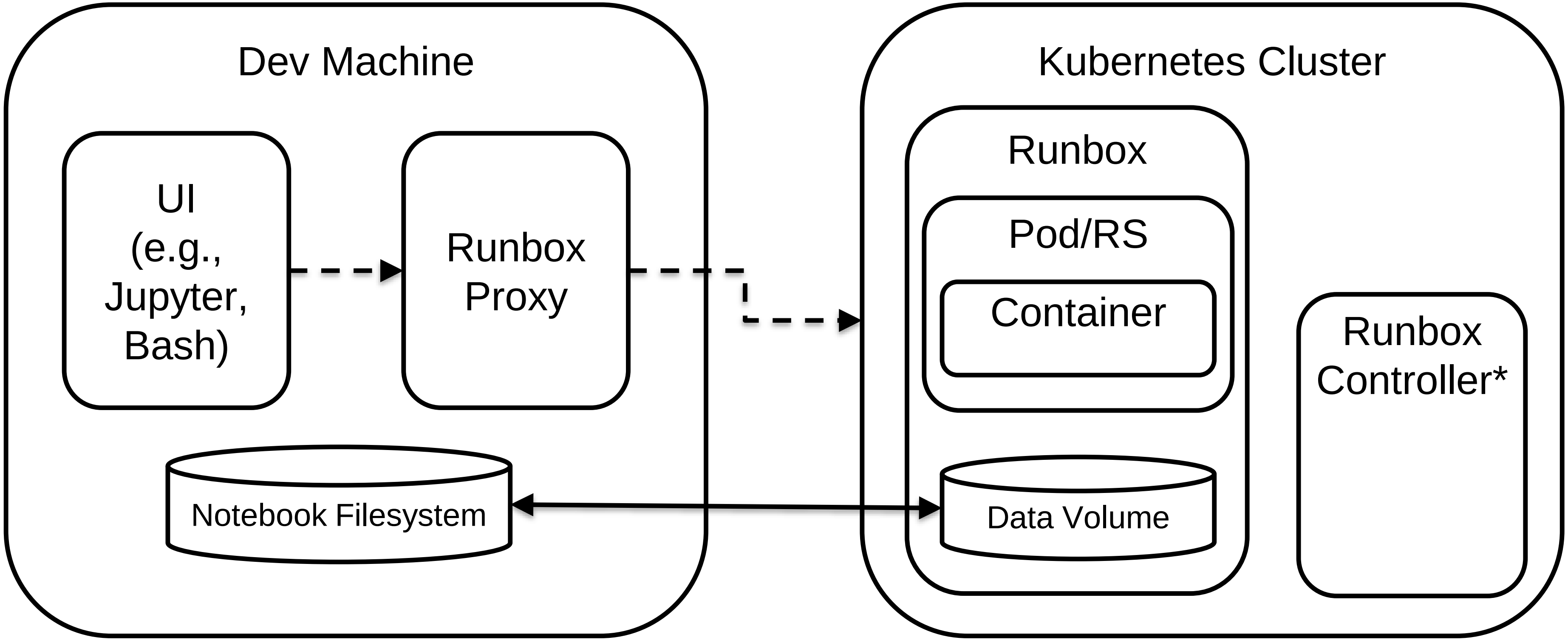
SCP for Interactive Computing

Property	Interactive Computing (Jupyter, Shell)
Code	Python, Bash
Application Pattern	Iterative invocation of stateful, non-parallel, computation-intensive, ad-hoc tasks, triggered by explicit user interaction
Management	Provisioning, management, scaling of underlying resources
Integration	Data sources, auth, etc
Elasticity, Pay-per-use	

SCP for Interactive Computing

Property	Interactive Computing (Jupyter, Shell)
Code	Python, Bash
Application Pattern	Iterative invocation of stateful, non-parallel, computation-intensive, ad-hoc tasks, triggered by explicit user interaction
Management	Provisioning, management, scaling of underlying resources
Integration	Data sources, auth, etc
Elasticity, Pay-per-use	Flexible resource allocation (vertical scaling) guided by user input (e.g., magics); Scale to zero when idle

Runbox: Elastic Persistent Execution Environment on K8s



DEMO – Bash

gliksonatpglikson: ~/kubecon19

```
glikson@tpglikson:~/kubecon19$ runbox --create myrb --image ubuntu:18.10
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                          READY   STATUS    RESTARTS   AGE
pod/myrb-6d5dd457c6-bhfjp     2/2     Running   0          22s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/myrb     1/1     1            1           22s
glikson@tpglikson:~/kubecon19$ runbox myrb hostname
myrb
```

Runbox environment:
Pod, Image, Volume,
(+deployment, side-car)

Remote command execution

Filesystem synchronization

```
glikson@tpglikson: ~/kubecon19
glikson@tpglikson:~/kubecon19$ runbox --create myrb --image ubuntu:18.10
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                         READY   STATUS    RESTARTS   AGE
pod/myrb-6d5dd457c6-bhfjp    2/2     Running   0          22s

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/myrb    1/1     1            1           22s
glikson@tpglikson:~/kubecon19$ runbox myrb hostname
myrb
glikson@tpglikson:~/kubecon19$ ls
my_file
glikson@tpglikson:~/kubecon19$ runbox --sync_before --localpath . myrb ls /data
my_file
```

Runbox environment: Pod, Image, Volume, (+deployment, side-car)

Remote command execution

Filesystem synchronization

Persistent over recycling of idle resource (e.g., by Runbox controller)

glikson@tpglikson: ~/kubecon19

```
glikson@tpglikson:~/kubecon19$ runbox --create myrb --image ubuntu:18.10
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                         READY   STATUS    RESTARTS   AGE
pod/myrb-6d5dd457c6-bhfjp    2/2     Running   0          22s

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/myrb    1/1     1            1           22s
glikson@tpglikson:~/kubecon19$ runbox myrb hostname
myrb
glikson@tpglikson:~/kubecon19$ ls
my_file
glikson@tpglikson:~/kubecon19$ runbox --sync_before --localpath . myrb ls /data
my_file
glikson@tpglikson:~/kubecon19$ kubectl scale --replicas 0 deployment myrb
deployment.extensions/myrb scaled
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/myrb    0/0     0            0           115s
glikson@tpglikson:~/kubecon19$ runbox myrb ls /data
my_file
```

Runbox environment: Pod, Image, Volume, (+deployment, side-car)

Remote command execution

Filesystem synchronization

Persistent over recycling of idle resource (e.g., by Runbox controller)

Per-command vertical scaling

glikson@tpglikson: ~/kubecon19

```
glikson@tpglikson:~/kubecon19$ runbox --create myrb --image ubuntu:18.10
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                        READY   STATUS    RESTARTS   AGE
pod/myrb-6d5dd457c6-bhfjp   2/2     Running   0          22s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/myrb   1/1     1            1           22s
glikson@tpglikson:~/kubecon19$ runbox myrb hostname
myrb
glikson@tpglikson:~/kubecon19$ ls
my_file
glikson@tpglikson:~/kubecon19$ runbox --sync_before --localpath . myrb ls /data
my_file
glikson@tpglikson:~/kubecon19$ kubectl scale --replicas 0 deployment myrb
deployment.extensions/myrb scaled
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/myrb   0/0     0            0           115s
glikson@tpglikson:~/kubecon19$ runbox myrb ls /data
my_file
glikson@tpglikson:~/kubecon19$ runbox myrb cat /sys/fs/cgroup/memory/memory.limit_in_bytes
134217728
glikson@tpglikson:~/kubecon19$ runbox -a 2 myrb cat /sys/fs/cgroup/memory/memory.limit_in_bytes
268435456
glikson@tpglikson:~/kubecon19$
```

Runbox environment: Pod, Image, Volume, (+deployment, side-car)

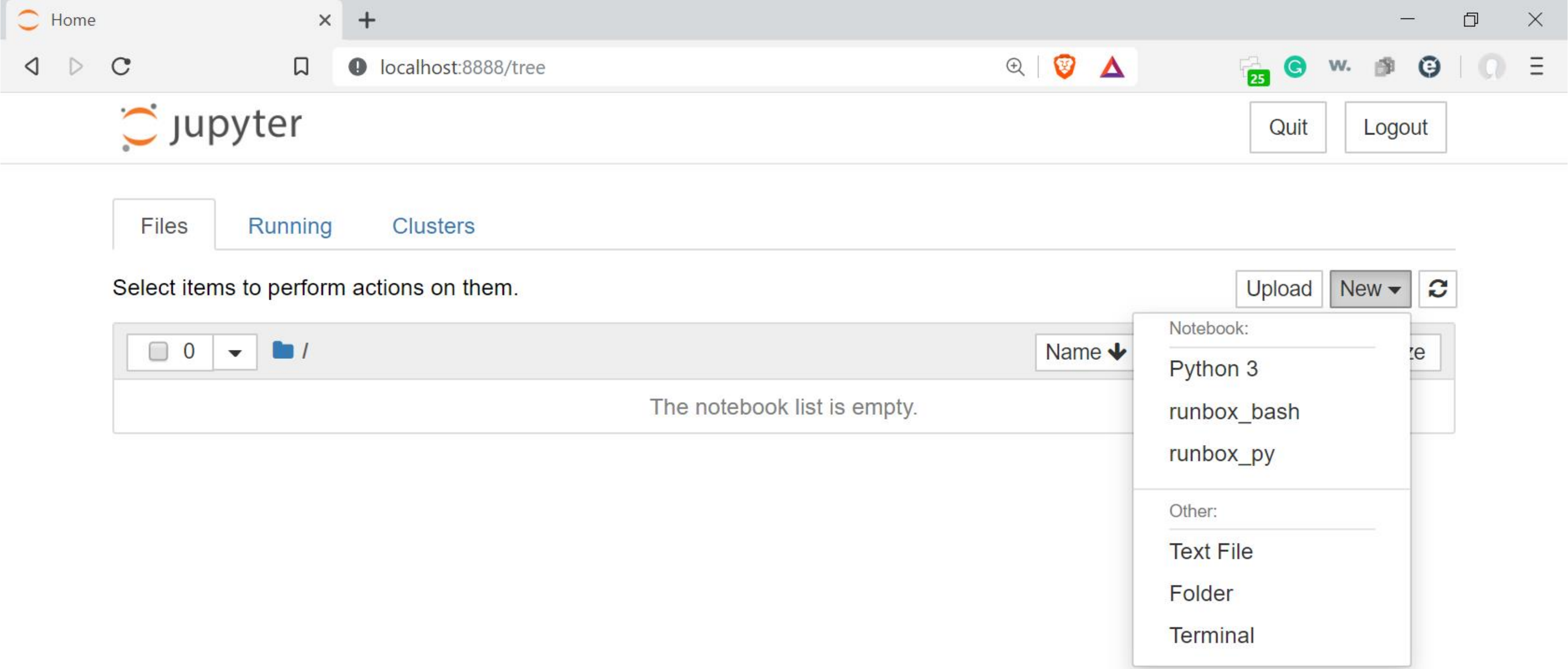
Remote command execution

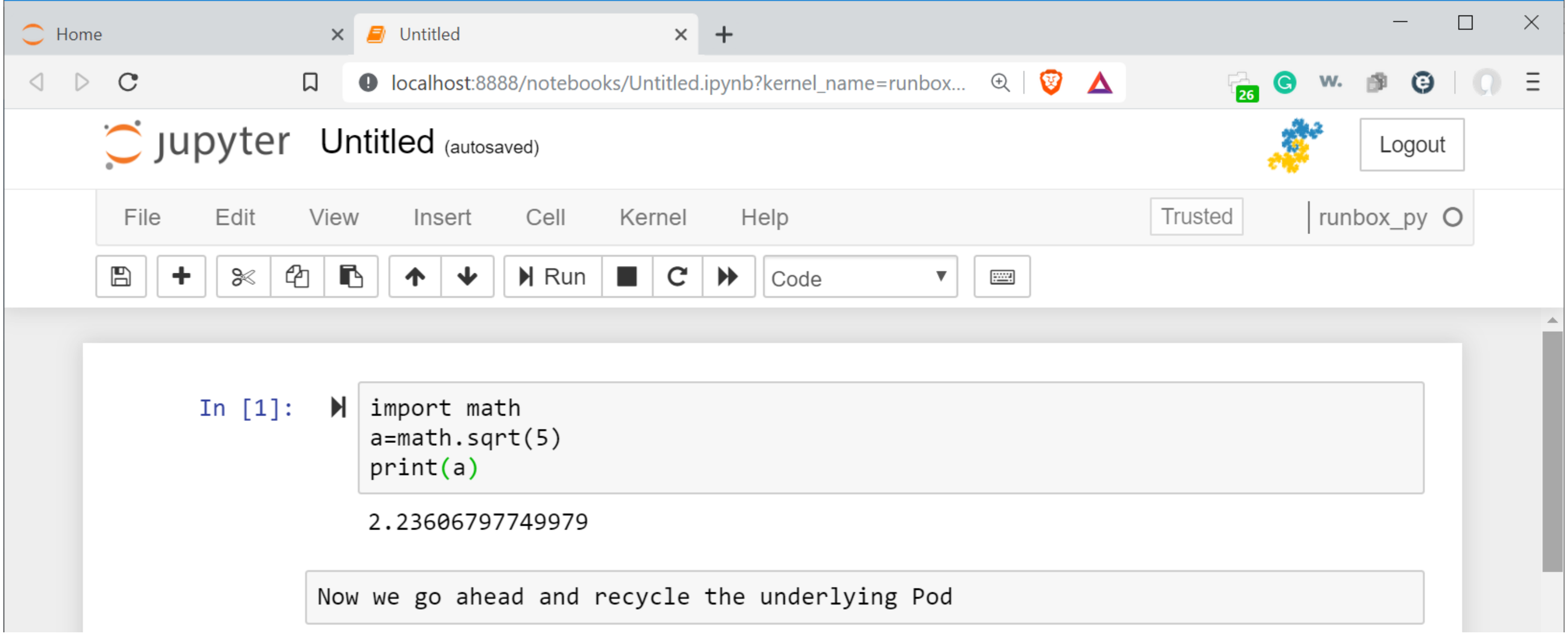
Filesystem synchronization

Persistent over recycling of idle resource (e.g., by Runbox controller)

Per-command vertical scaling

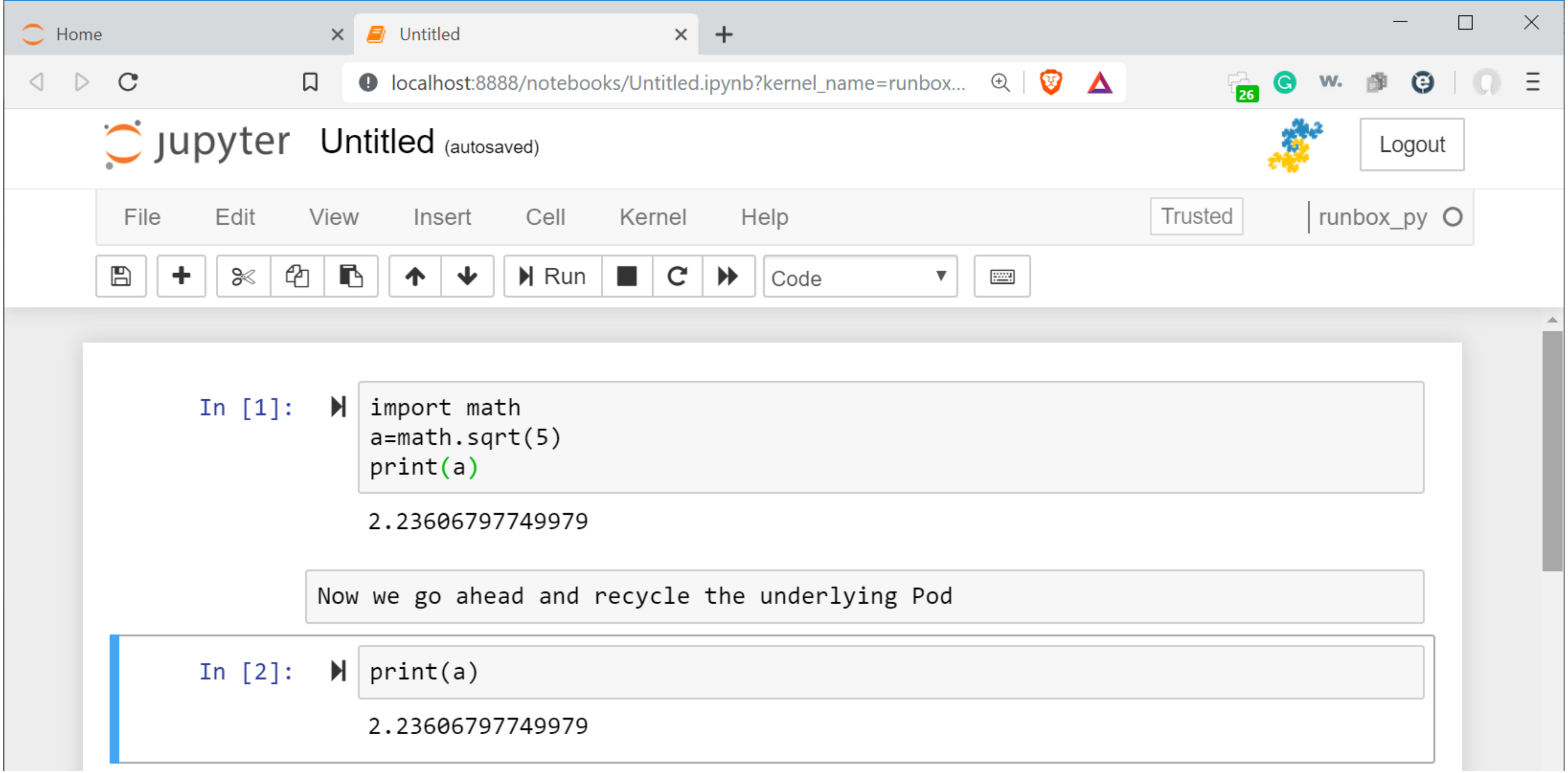
DEMO – Jupyter

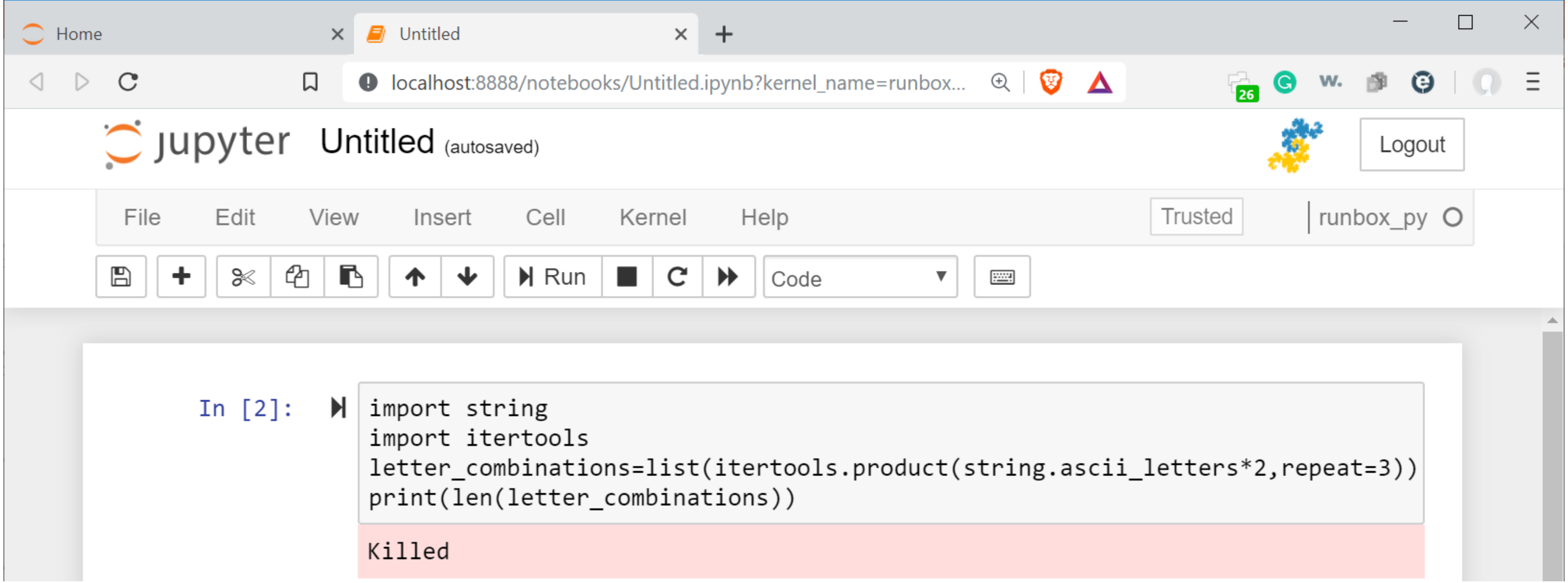

Home
localhost:8888/tree
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
The notebook list is empty.
Notebook:
Python 3
runbox_bash
runbox_py
Other:
Text File
Folder
Terminal

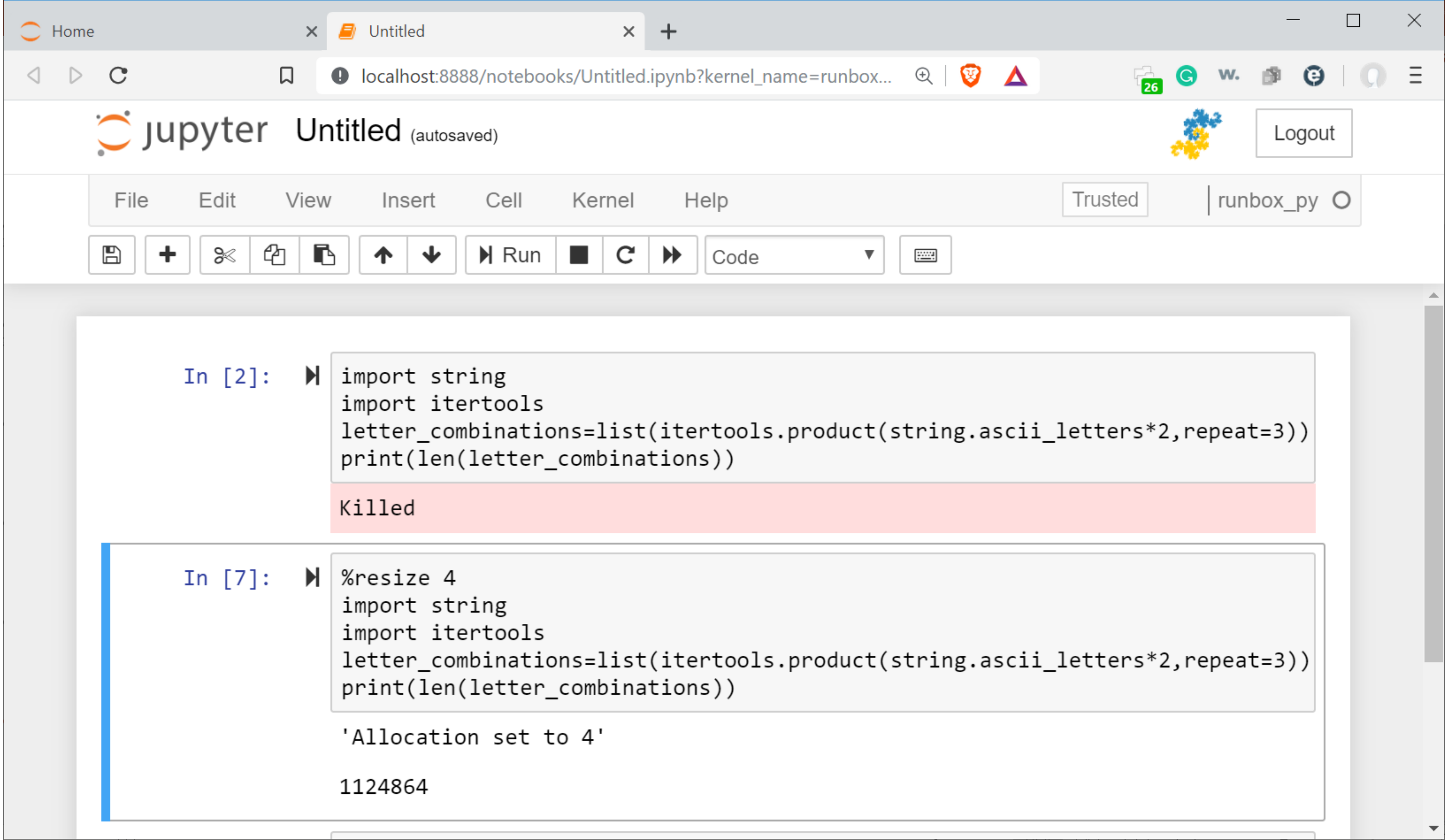

Home
Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=runbox...
jupyter
Untitled (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
runbox_py
Run
Code
In [1]:
import math
a=math.sqrt(5)
print(a)
2.23606797749979
Now we go ahead and recycle the underlying Pod

glikson@tpglikson: ~/kubecon19

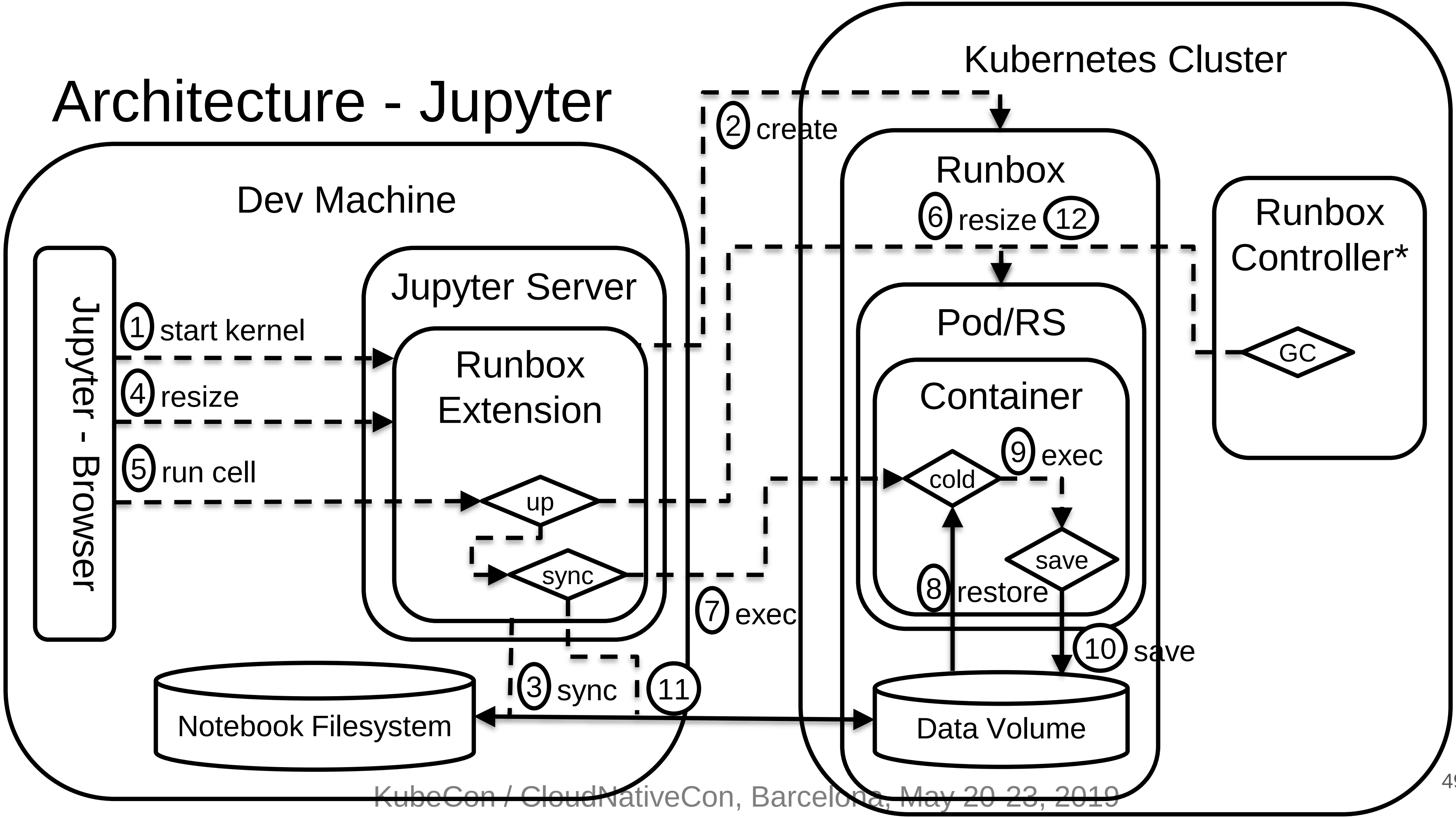
```
glikson@tpglikson:~/kubecon19$ runbox --list
ipy-84
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                           READY   STATUS    RESTARTS   AGE
pod/ipy-84-8968fb5ff-hx6vm     2/2     Running   0          59s

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/ipy-84     1/1     1            1           59s
glikson@tpglikson:~/kubecon19$ runbox --kill ipy-84 true
glikson@tpglikson:~/kubecon19$ kubectl get "pod,deployment"
NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/ipy-84     0/0     0            0           107s
glikson@tpglikson:~/kubecon19$
```


Home
Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=runbox...
jupyter Untitled (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
runbox_py
Run
Code
In [1]:
import math
a=math.sqrt(5)
print(a)
2.23606797749979
Now we go ahead and recycle the underlying Pod
In [2]:
print(a)
2.23606797749979


Home
Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=runbox...
jupyter
Untitled (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
runbox_py
Run
Code
In [2]:
import string
import itertools
letter_combinations=list(itertools.product(string.ascii_letters*2,repeat=3))
print(len(letter_combinations))
Killed


Home
Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=runbox...
jupyter
Untitled (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
runbox_py
Run
Code
In [2]:
import string
import itertools
letter_combinations=list(itertools.product(string.ascii_letters*2,repeat=3))
print(len(letter_combinations))
Killed
In [7]:
%resize 4
import string
import itertools
letter_combinations=list(itertools.product(string.ascii_letters*2,repeat=3))
print(len(letter_combinations))
'Allocation set to 4'
1124864

Architecture - Jupyter



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

Lessons Learned

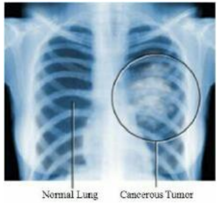
- Kubernetes originally focused on scale-out workloads, but can also support scale-up
    - New kind of controller?
- Generic support for application-assisted snapshots could be useful
- For use-cases involving ephemeral compute, API for direct access to volumes could be useful

Outline

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

Deep Learning

transportation



medicine

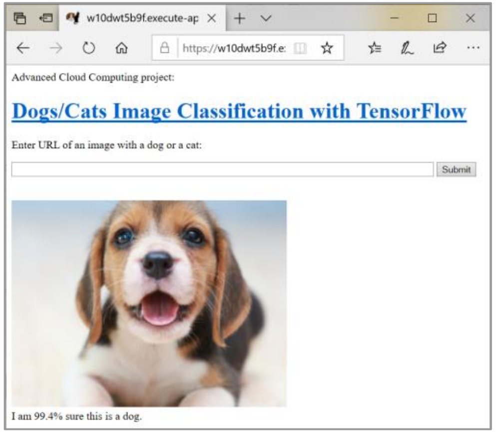

smart cities, security


consumer

games

e-commerce

- Resource-intensive
  - (1) model training, (2) inference
- Frameworks: Tensorflow, Keras, PyTorch, etc.
  - ‘Hot’ research area – new algorithms, frameworks, etc
- Example application: Image Classification
  - Given a model + unlabeled example(s), predict label(s)
  - Compute-intensive, scale-out, can leverage GPUs



SCP for Deep Learning Inference

Property	Deep Learning Inference
Code	
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

SCP for Deep Learning Inference

Property	Deep Learning Inference
Code	Model inference implementation (Python)
Application Pattern	
Management	
Integration	
Elasticity, Pay-per-use	

SCP for Deep Learning Inference

Property	Deep Learning Inference
Code	Model inference implementation (Python)
Application Pattern	Long-running, scale-out services; Linear resource demand per request; Load variance Can benefit from running on GPUs; potentially large “cold-start” latencies
Management	
Integration	
Elasticity, Pay-per-use	

SCP for Deep Learning Inference

Property	Deep Learning Inference
Code	Model inference implementation (Python)
Application Pattern	Long-running, scale-out services; Linear resource demand per request; Load variance Can benefit from running on GPUs; potentially large “cold-start” latencies
Management	Same as Knative: build, serving, eventing Load balancing between GPU and CPU resources; Minimal ‘cold-start’ latency
Integration	
Elasticity, Pay-per-use	

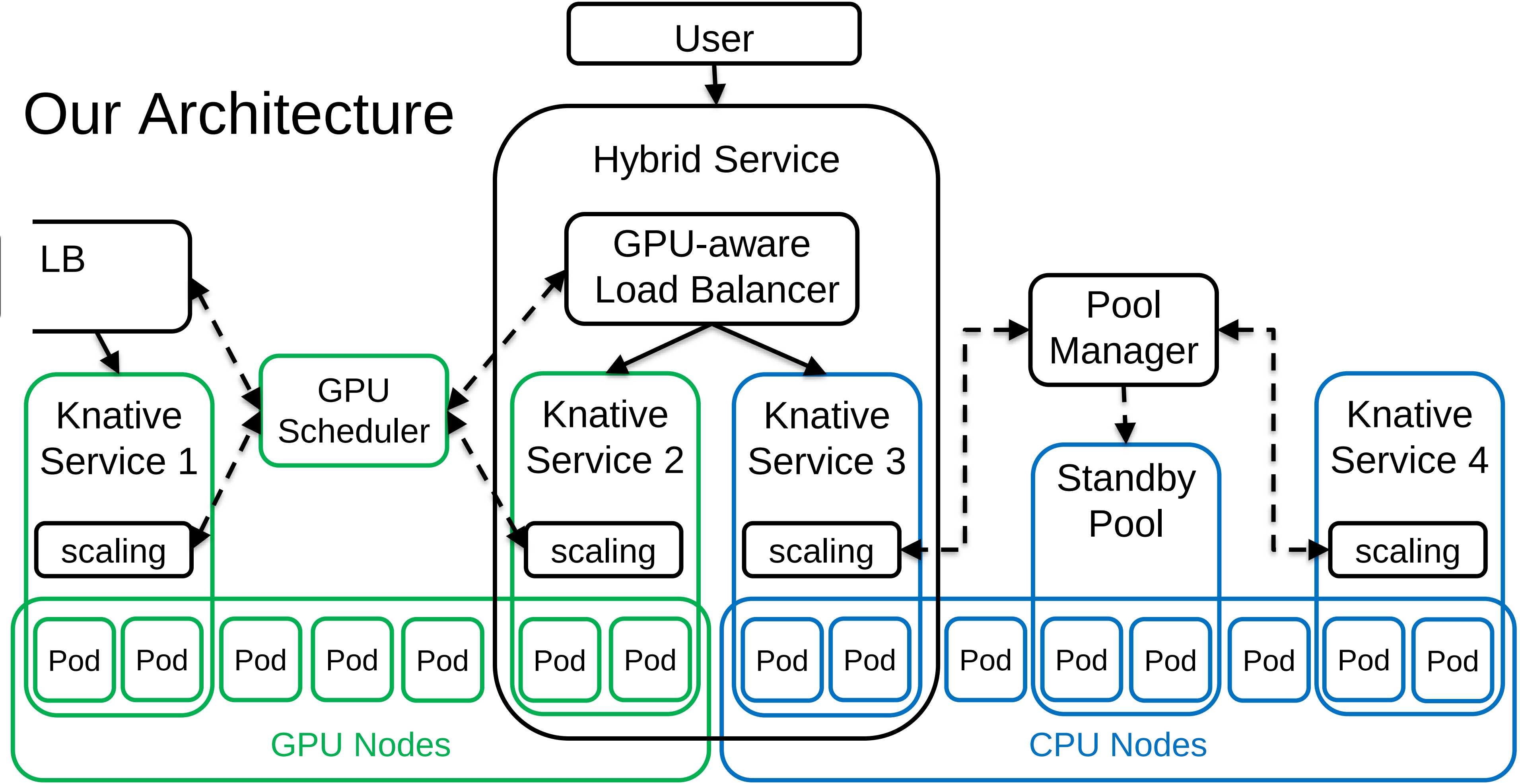
SCP for Deep Learning Inference

Property	Deep Learning Inference
Code	Model inference implementation (Python)
Application Pattern	Long-running, scale-out services; Linear resource demand per request; Load variance Can benefit from running on GPUs; potentially large “cold-start” latencies
Management	Same as Knative: build, serving, eventing Load balancing between GPU and CPU resources; Minimal ‘cold-start’ latency
Integration	K8s, Istio, model storage, etc
Elasticity, Pay-per-use	

SCP for Deep Learning Inference

Property	Deep Learning Inference
Code	Model inference implementation (Python)
Application Pattern	Long-running, scale-out services; Linear resource demand per request; Load variance Can benefit from running on GPUs; potentially large “cold-start” latencies
Management	Same as Knative: build, serving, eventing Load balancing between GPU and CPU resources; Minimal ‘cold-start’ latency
Integration	K8s, Istio, model storage, etc
Elasticity, Pay-per-use	Request-based scaling, 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
- **Conclusions**

Conclusions

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

Questions? Ideas? Suggestions?

- alex.glikson at gmail.com