MLDep: A Testbed for Performance Exploration of Machine Learning Inference Serving

Srishankar Sundar | s3151298

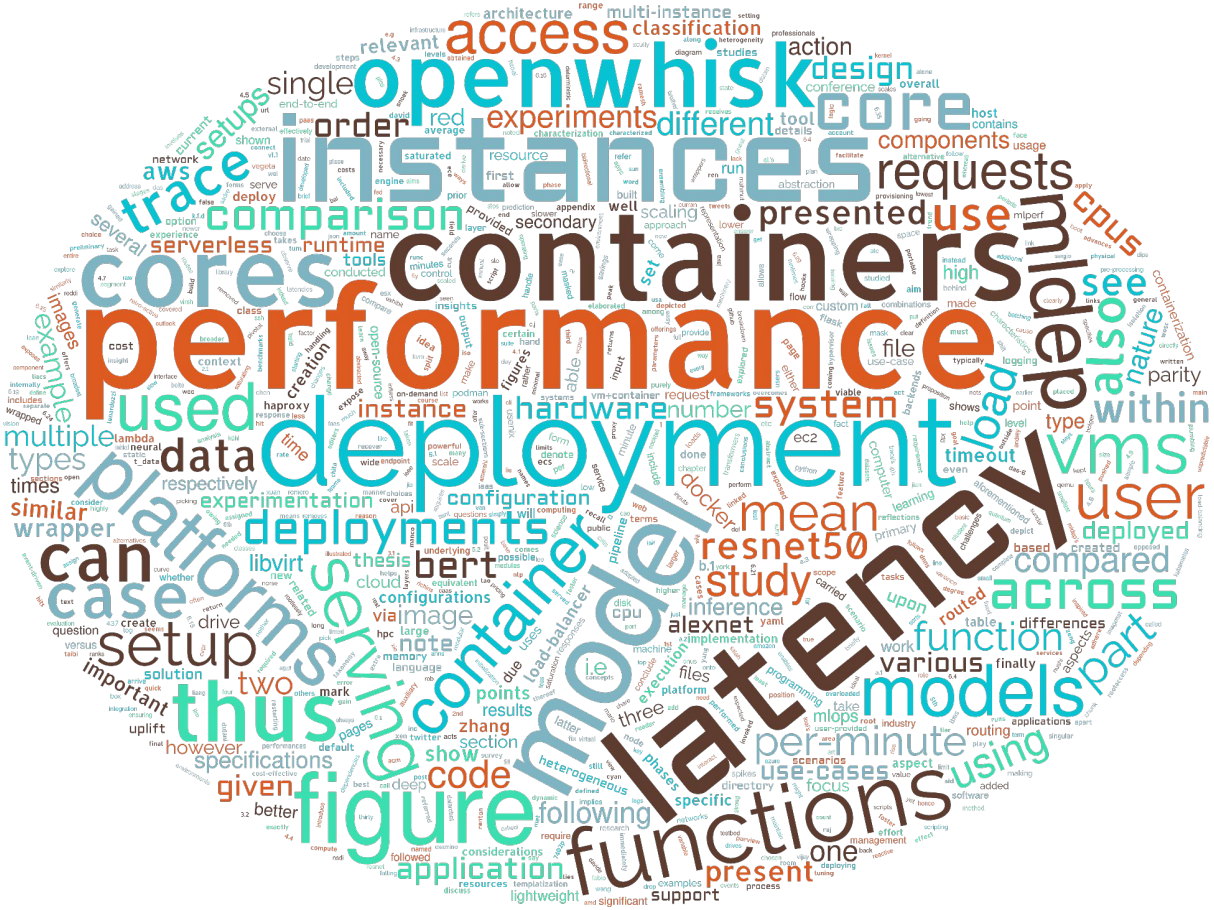
Universiteit Leiden
The Netherlands

1st Supervisor: Dr. K.F.D. Rietveld

2nd Reader: Prof. Dr. Rob van Nieuwpoort

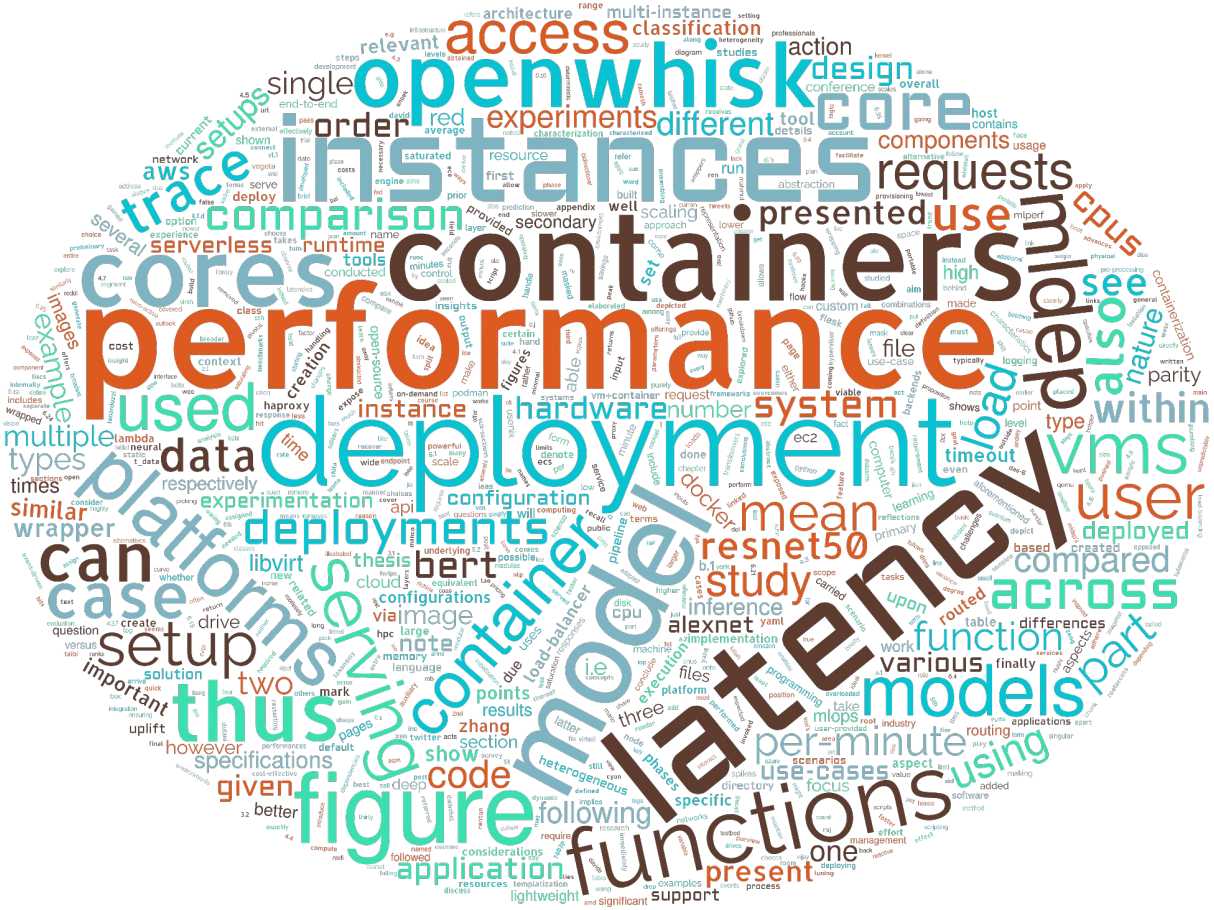
Overview

- Introduction
- Research Questions
- MLDep - Design and Implementation
- Experimentation Approach
- Results
- Conclusion
- Future Work



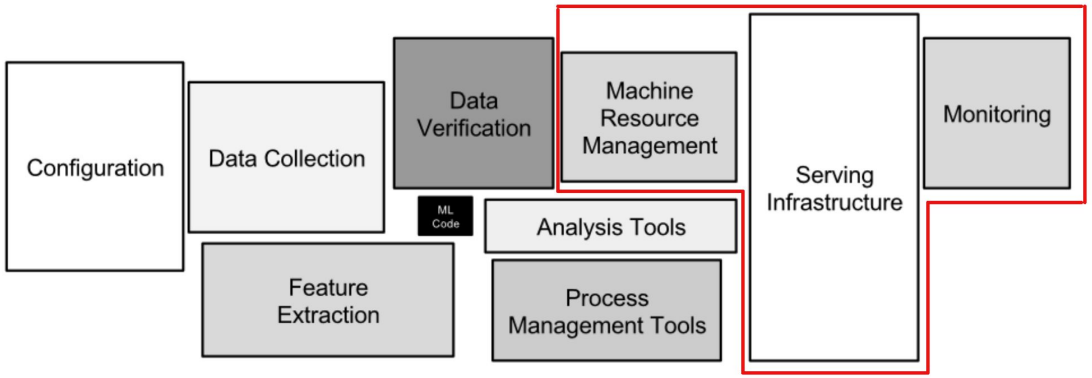
Overview

- **Introduction**
- Research Questions
- MLDep - Design and Implementation
- Experimentation Approach
- Results
- Conclusion
- Future Work


performance
latency
containers
deployment
instances
openwhisk
mldep
models
functions
figure
thus
cores
platforms
serverless
deployments
model
container
setup
case
can
data
used
trace
comparison
access
core
requests
vms
user
mean
resnet50
study
across
compared
function
per-minute
using
code
application
present
one
support
experiments
different
components
presented
use
cpus
see
also
within
system
hardware
instance
configuration
experimentation
bert
libvirt
image
configurations
alexnet
various
part
multiple
types
similar
wrapper
times
aws
setups
single
order
red
relevant
classification
multi-instance
architecture
action
design
deployed
timeout
load
nature
containerization
parity
type
point
shows
high
file
use-case
runtime
tools
secondary
scaling
well
resource
first
run
abstraction
provisioning
network
deploy
serve
several
images
example
open-source
creation
haproxy
cost
context
number
ec2
docker
computer
learning
reflections
primary
terms
pipeline
execution
load-balancer
uses
cpu
disk
inference
upon
routed
table
differences
work
aspects
routing
applications
apart
mlops
take
scenarios
aspect
specific
directory
focus
software
added
management
effort
deploying
tuning
process
examples
templatization
resources
lightweight
significant
considerations
followed
heterogeneous
phases
platform
programming
machine
machines
cloud
equivalent
related
new
whether
drive
question
versus
important
solution
uplift
however
specifications
given
better
deep
card
section
show
zhang
results
points
three
note
memory
language
hpc
large
pages
twitter
acts
default
mark
two
solution
due

Introduction

- Inference serving
- Deployment
  - MLOps tools
  - IaaS, CaaS, FaaS
- MLDep
  - Fewer layers of abstraction
  - Light-weight, modular and extensible
  - Focus on comparison of deployment platforms
- Study on nature of deployments



Scully et al.

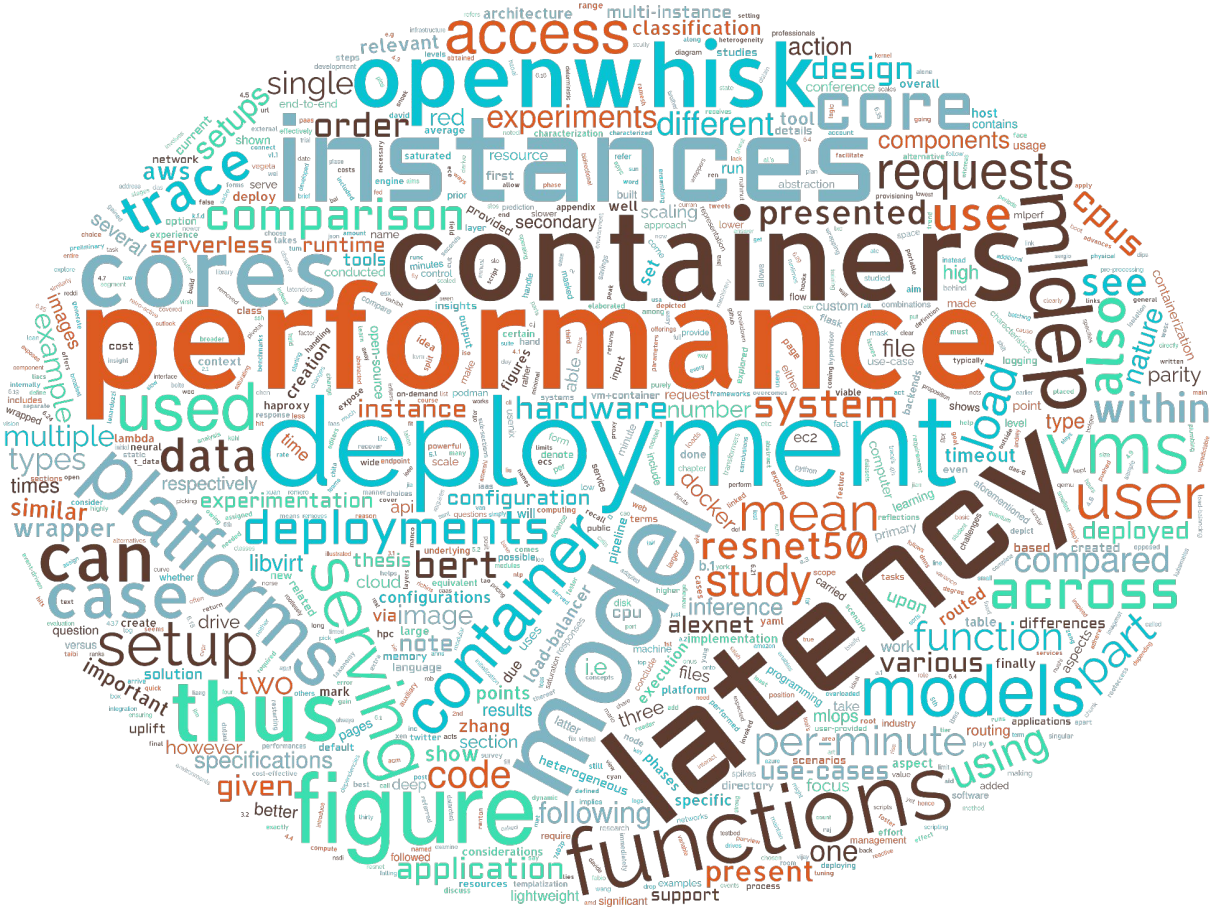
Amazon SageMaker

Kubeflow

H2O.ai

Overview

- Introduction
- **Research Questions**
- MLDep - Design and Implementation
- Experimentation Approach
- Results
- Conclusion
- Future Work



Research Questions

- What are the **potential challenges and pitfalls** associated with different deployment platforms?
- What are the **comparative performance results of the underlying deployment platforms/configurations** for currently relevant ML use-cases?

Contributions:

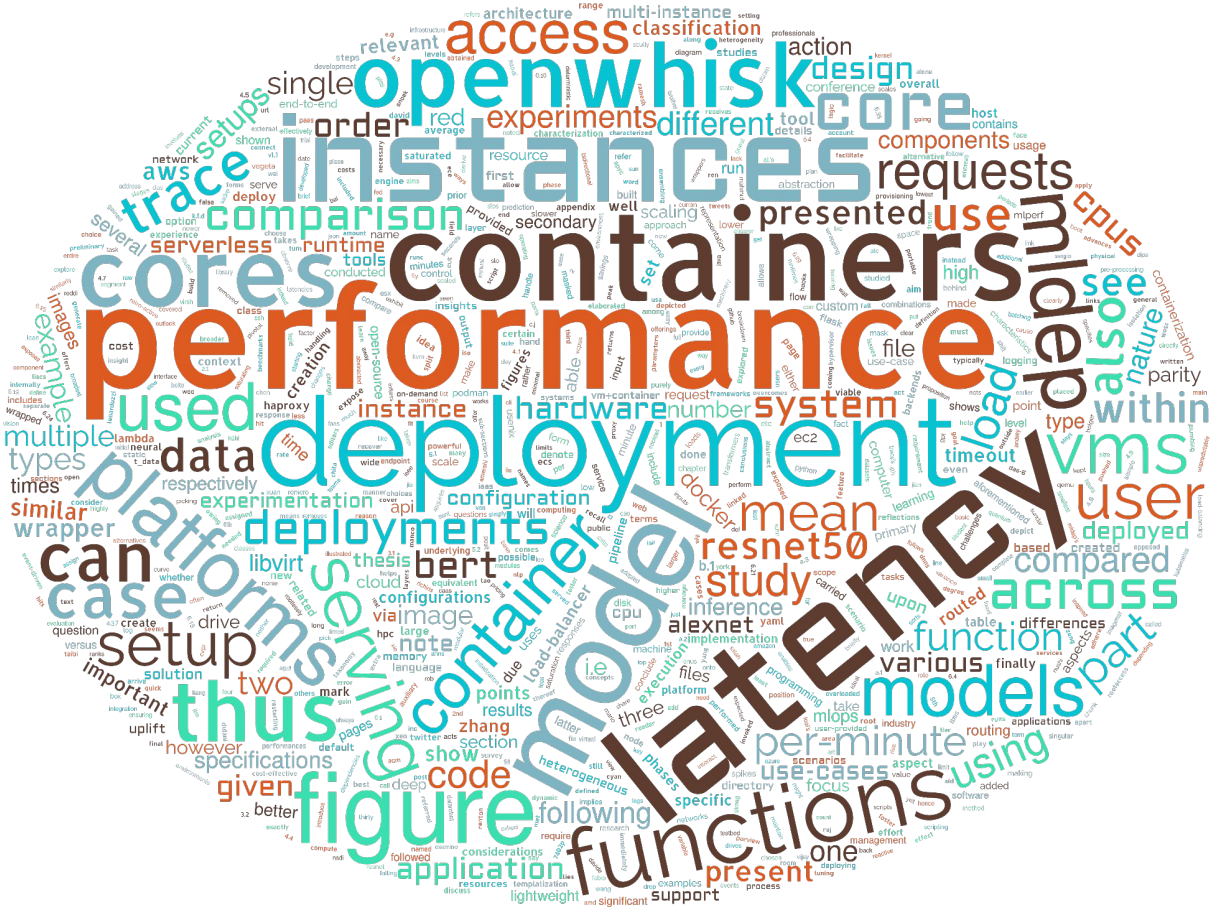
- Multiple deployment setups tested
- Hardware parity between platforms
- **MLDep** - Fosters **exploration** of deployment characteristics given a model

ML Model	EC2		ECS		Lambda	
	$	t (ms)	$	t (ms)	$	t (ms)
Inception-v3	5.0	210	9.17	217	19.0	380
Inception-ResNet	9.3	398	16.4	411	39.3	785
OpenNMT-ende	51.5	2180	96.3	2280	155	3100

Zhang et al.

Overview

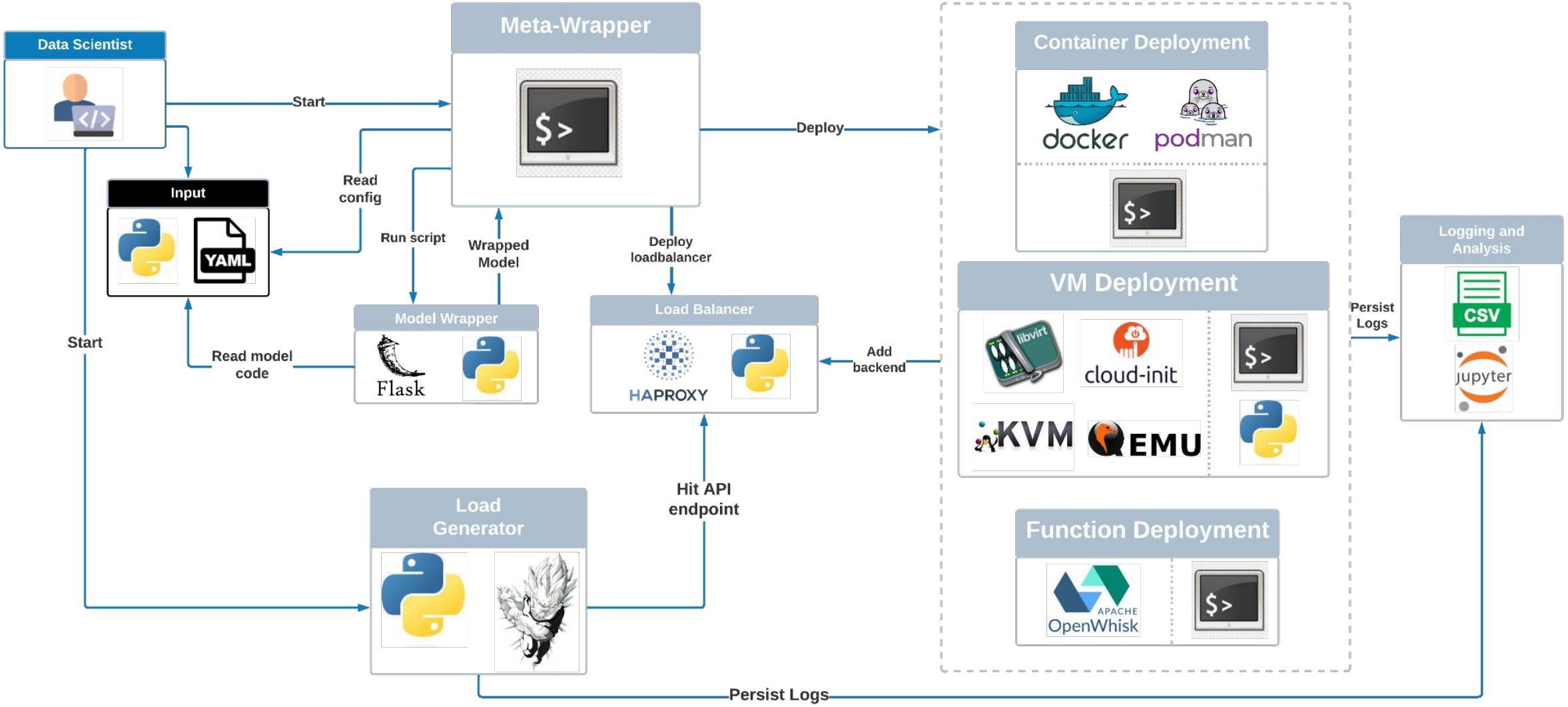
- Introduction
- Research Questions
- **MLDep - Design and Implementation**
- Experimentation Approach
- Results
- Conclusion
- Future Work


performance
latency
containers
deployment
instances
openwhisk
models
functions
figure
thus
setup
platforms
serverless
deployments
mean
resnet50
vms
user
across
core
requests
access
data
used
cores
trace
comparison
mldep
cpus
see
also
within
case
can
bert
study
function
part
using
per-minute
code
application
present
one
support
lightweight
container
model
system
hardware
instance
configuration
experimentation
multiple
types
similar
wrapper
times
images
example
setups
aws
single
order
experiments
different
presented
use
components
secondary
runtime
tools
conducted
haproxy
libvirt
cloud
image
configurations
alexnet
inference
various
differences
compared
deployed
timeout
parity
file
docker
load-balancer
execution
heterogeneous
specifications
given
however
important
two
three
points
results
zhang
section
show
use-cases
specific
following
considerations
resources
templatization
lightweight
significant
examples
process
multi-instance
classification
architecture
action
design
relevant
red
secondary
scaling
high
point
type
level
ec2
cost
context
creation
open-source
figure
number
done
scale
denote
ecs
lambda
neural
static
t_data
endpoint
wide
deploy
serve
network
connect
vgpu
aws
cpu
memory
machine
language
pipeline
terms
web
computer
learning
reflections
primary
based
created
appeared
tasks
carried
upon
routed
table
work
aspects
applications
routing
mlops
industry
scenarios
aspect
focus
directory
software
added
implies
dynamic
implementation
programming
platform
files
phases
pages
default
twitter
acts
section
deep
better
uplift
question
versus
drive
solution
mark
pc
large
note
via
cpu
disk
upon
routed
table
differences

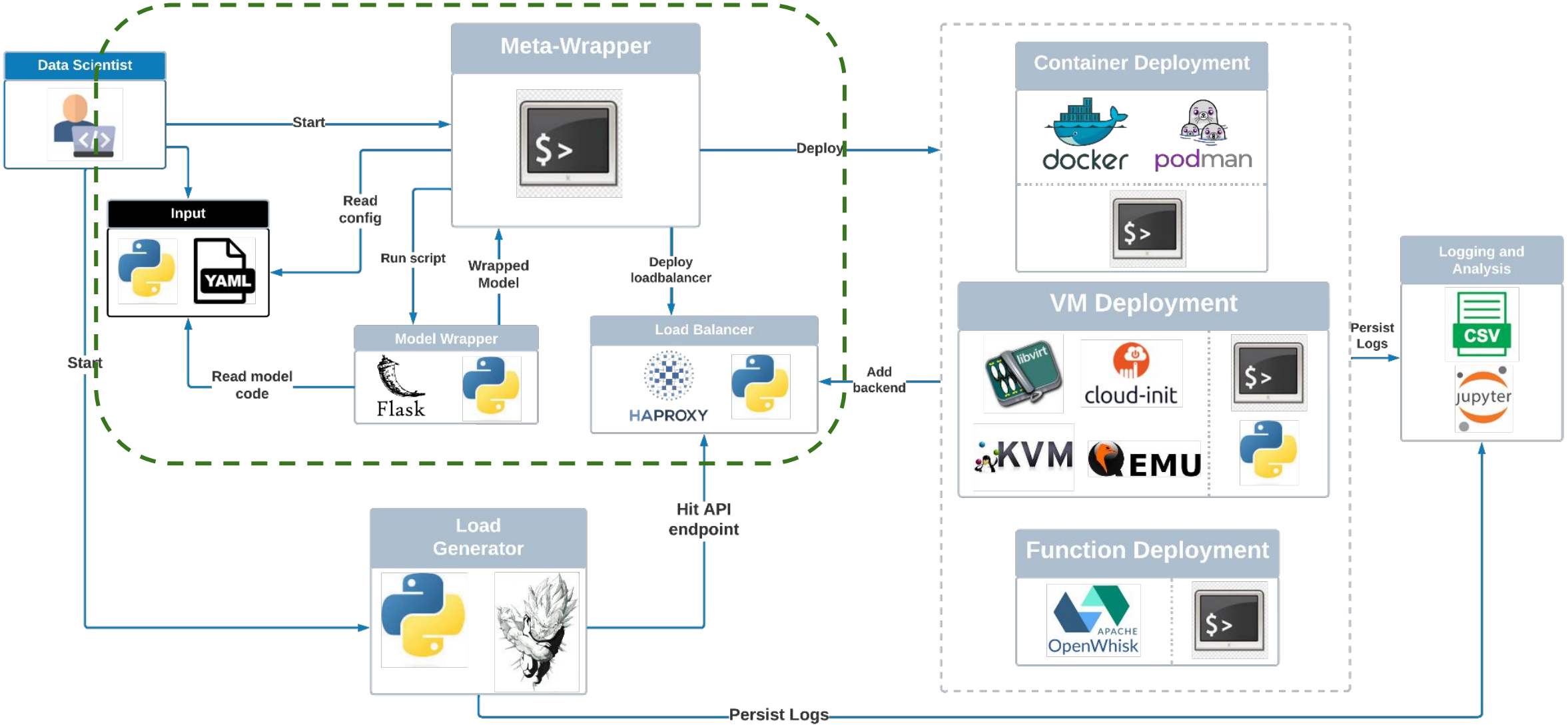
MLDep - Tech Stack

- Bash Script
- Jupyter Notebook
- Python Script or Model Code
- Podman
- cloud-init
- QEMU
- Flask
- HAProxy (Loadbalancer)
- Vegeta Load-testing
- Docker
- Libvirt
- KVM
- Apache OpenWhisk

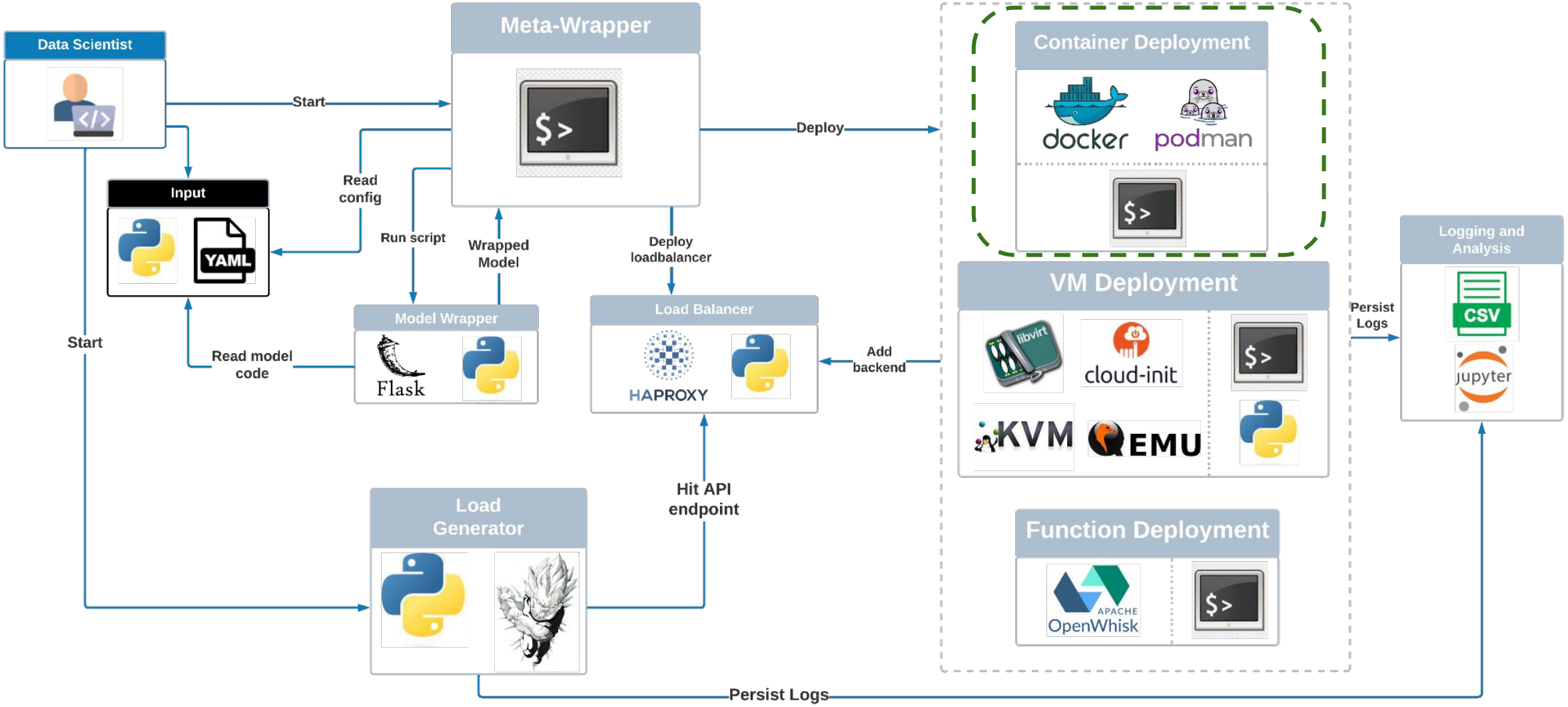
MLDep - Solution Architecture



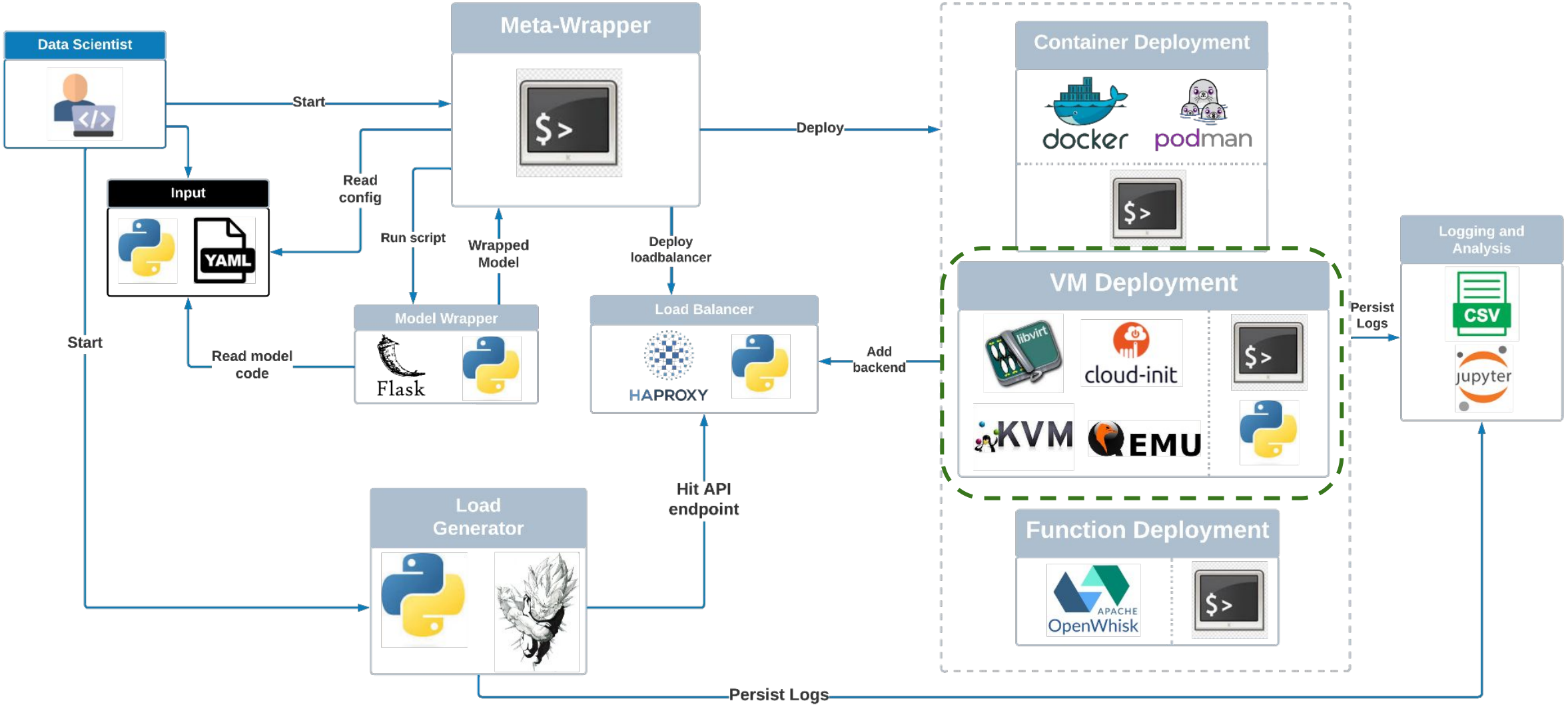
MLDep - Pre-deployment Phase



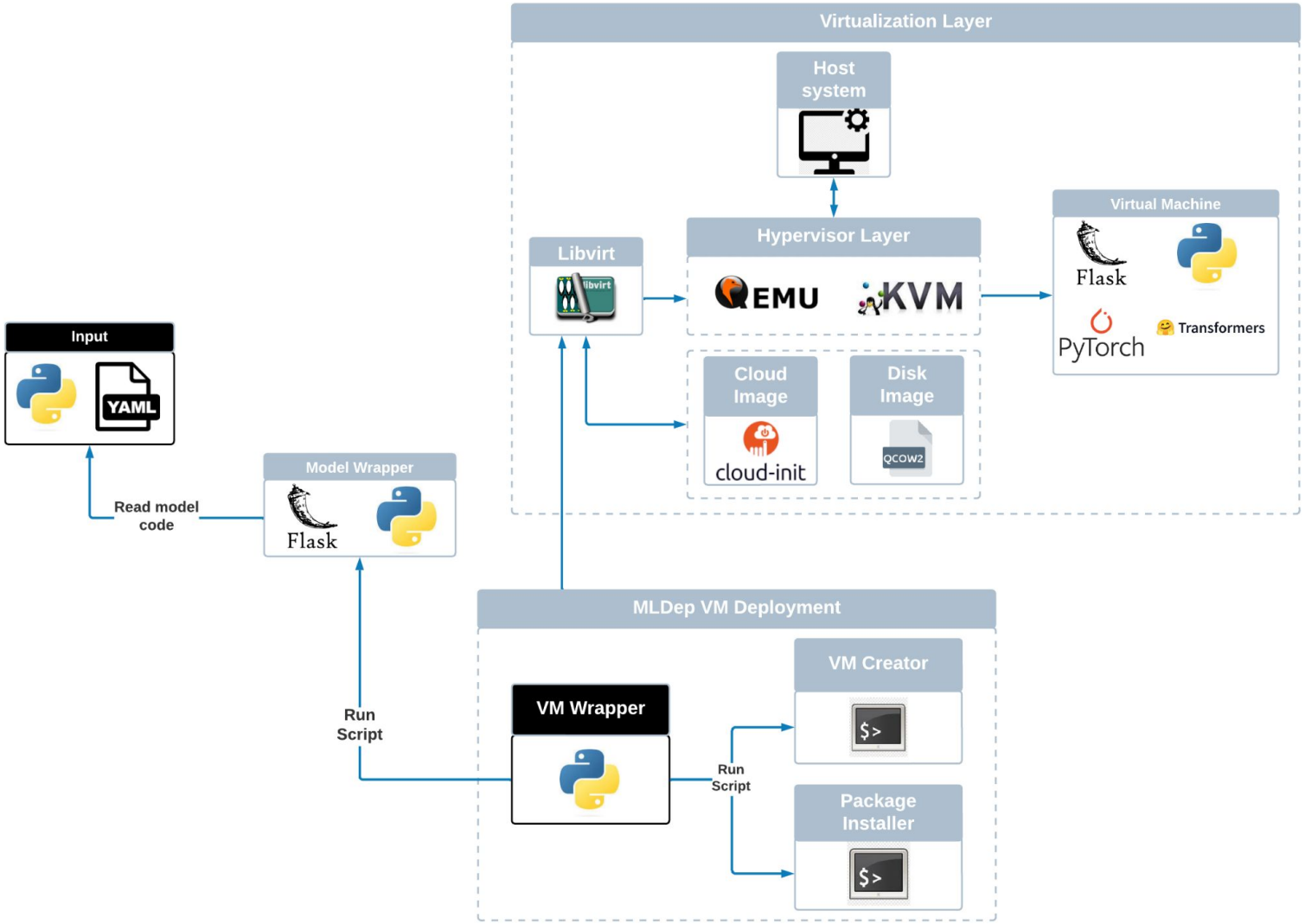
MLDep - Container Deployment



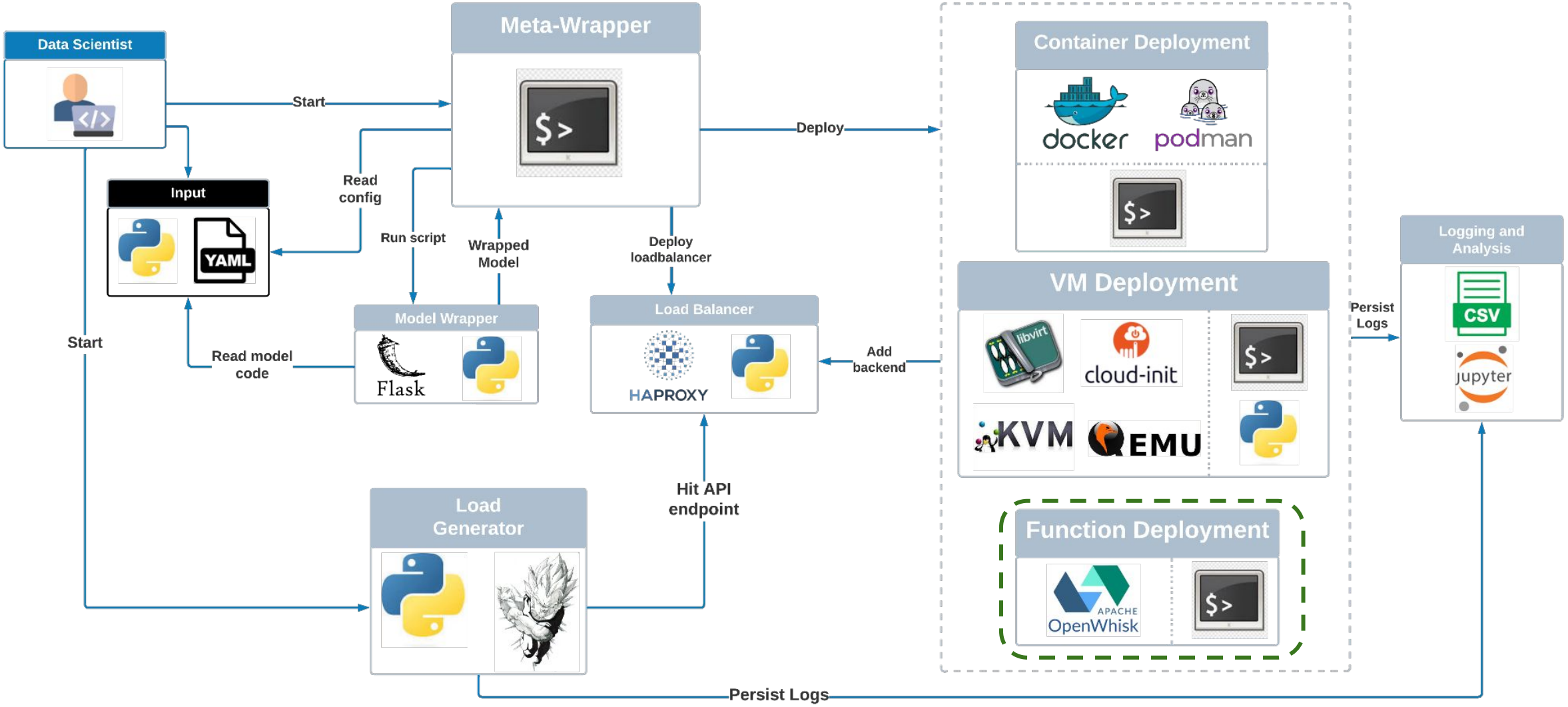
MLDep - VM Deployment



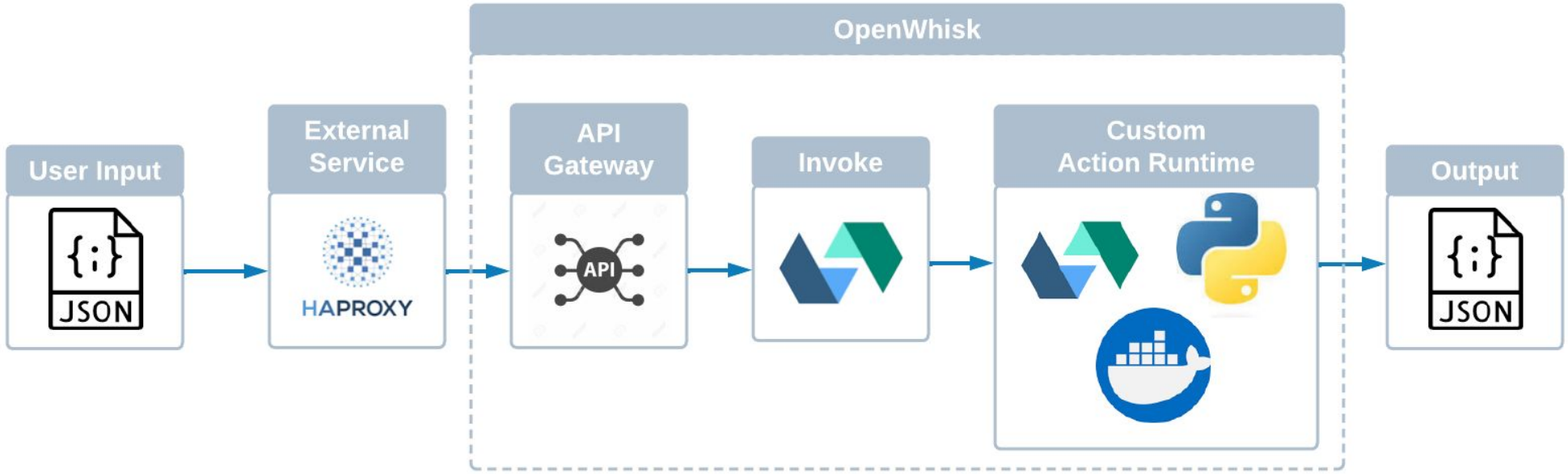
MLDep - VM Deployment



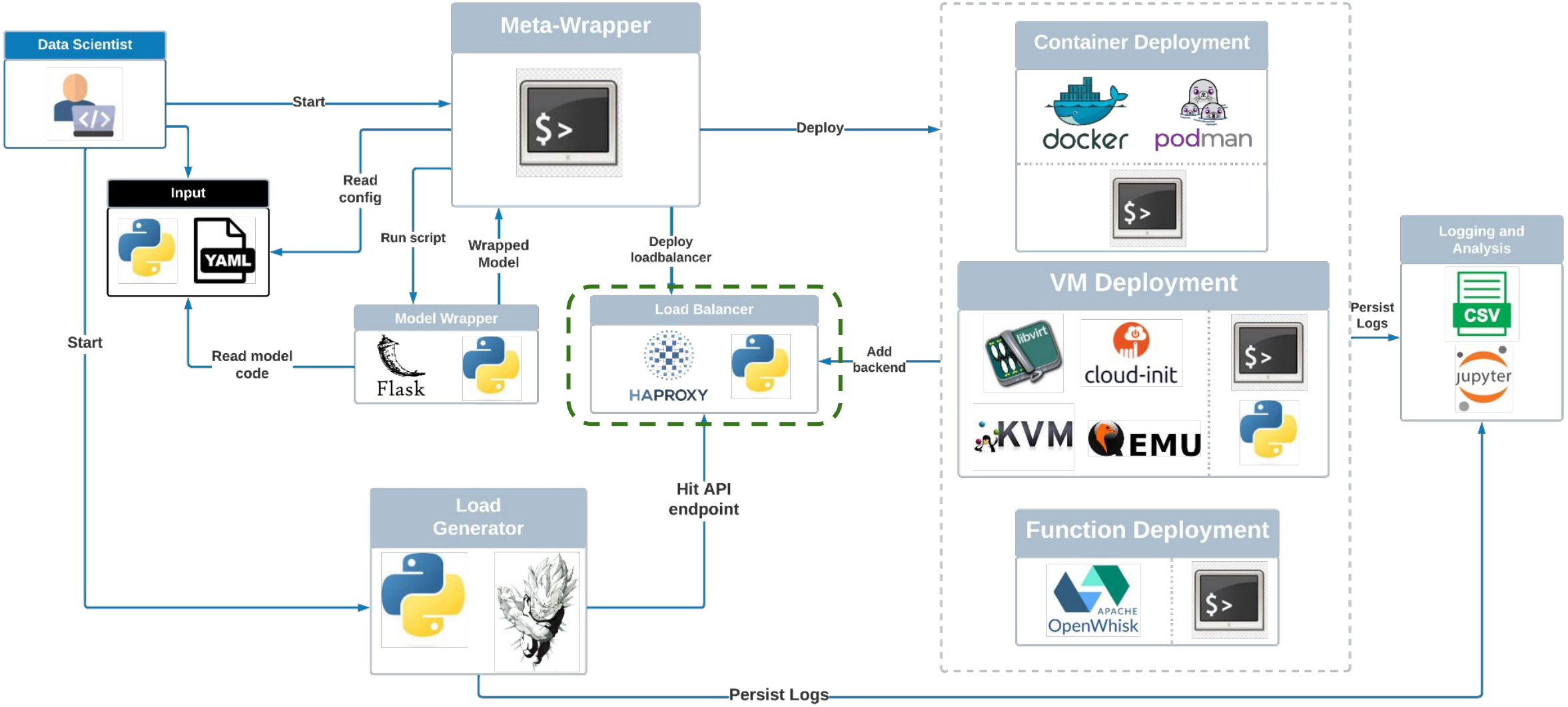
MLDep - Function Deployment



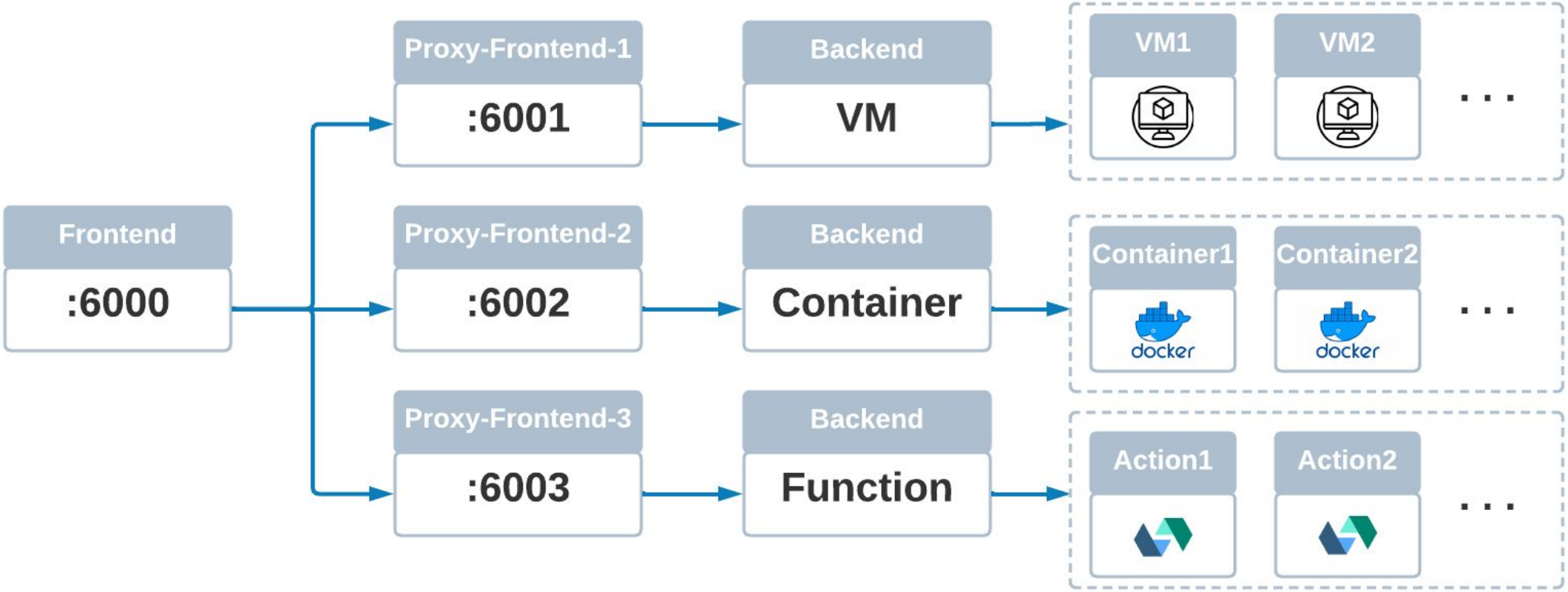
MLDep - OpenWhisk Functions



MLDep - Load-balancer

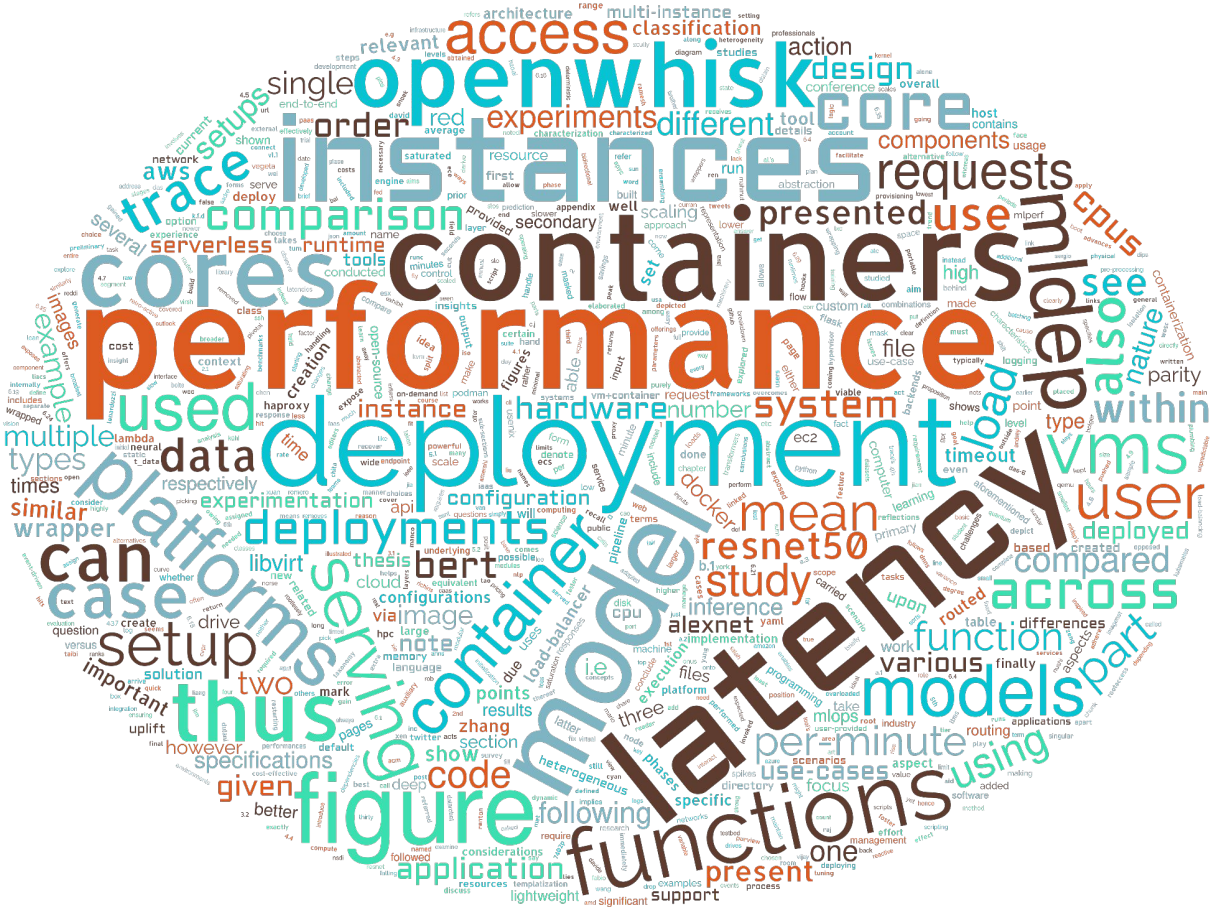


MLDep - Load-balancer



Overview

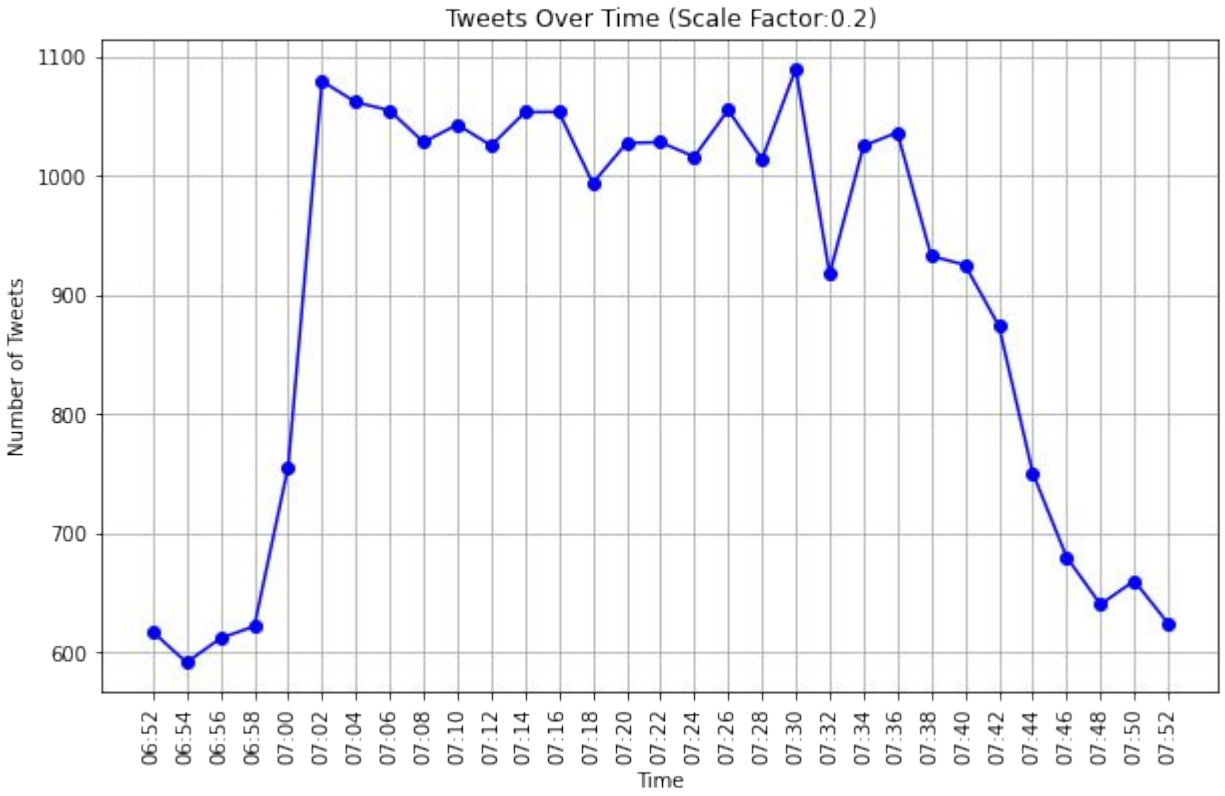
- Introduction
- Research Questions
- MLDep - Design and Implementation
- **Experimentation Approach**
- Results
- Conclusion
- Future Work


latency
performance
containers
deployment
instances
openwhisk
mldep
model
models
functions
figure
thus
cores
platforms
serving
container
case
can
setup
used
data
deployments
access
core
requests
vms
user
across
mean
resnet50
bert
study
function
trace
comparison
experiments
different
presented
use
cpus
see
also
within
deployed
compared
using
per-minute
use-cases
code
application
present
one
support
lightweight
given
specifications
however
two
important
similar
wrapper
multiple
types
times
experimentation
configuration
configurations
instance
hardware
system
serverless
runtime
tools
secondary
scaling
single
order
red
relevant
action
design
classification
multi-instance
architecture
components
timeout
docker
alexnet
inference
various
parity
containerization
nature
load
type
point
high
file
use-case
cost
context
creation
haproxy
libvirt
cloud
image
note
load-balancer
execution
heterogeneous
following
considerations
section
show
zhang
points
results
three
mlops
routing
aspect
specific
focus
scenarios
directory
software
added
deploying
tuning
effort
management
examples
process
significant
resources
templatization
discuss
deep
better
uplift
solution
drive
question
versus
pages
phases
twitter
acts
default
mark
aws
several
images
example
open-source
haproxy
response
less
lambda
neural
static
t_data
endpoint
wide
scale
denote
ecs
ec2
minute
time
service
pipeline
terms
computer
learning
reflections
primary
based
created
appeared
tasks
upon
routed
carried
differences
aspects
work
finally
applications
apart
routing
industry
mlperf
containerization
nature
parity
point
type
level
shows
made
mest
typically
logging
backends
viable
fact
flask
abstraction
provisioning
provided
appendix
built
well
slower
approach
lower
tweets
deployed
cost
hand
insights
conducted
control
minutes
name
tools
runtime
set
figure
figures
open-source
creation
context
cost
outlook
breaker
factor
handling
experience
option
takes
choice
deploy
serve
shown
network
current
connect
external
effectively
saturated
resource
first
prior
engine
aims
costs
forms
tweets
run
abstract
built
well
appendix
provisioning
provided
slower
approach
lower
scaling
secondary
runtime
tools
conducted
control
minutes
name

Models and Trace

Model	Use-Case	Framework/ Library	Parameters
AlexNet	Image Classification	PyTorch	62.3M
ResNet50	Image Classification	PyTorch	23.5M
BERT	Bi-directional Context Inference	Transformers	345M

- Load generation with Vegeta
- No repeated requests, uniform order across experiments
- Three phases

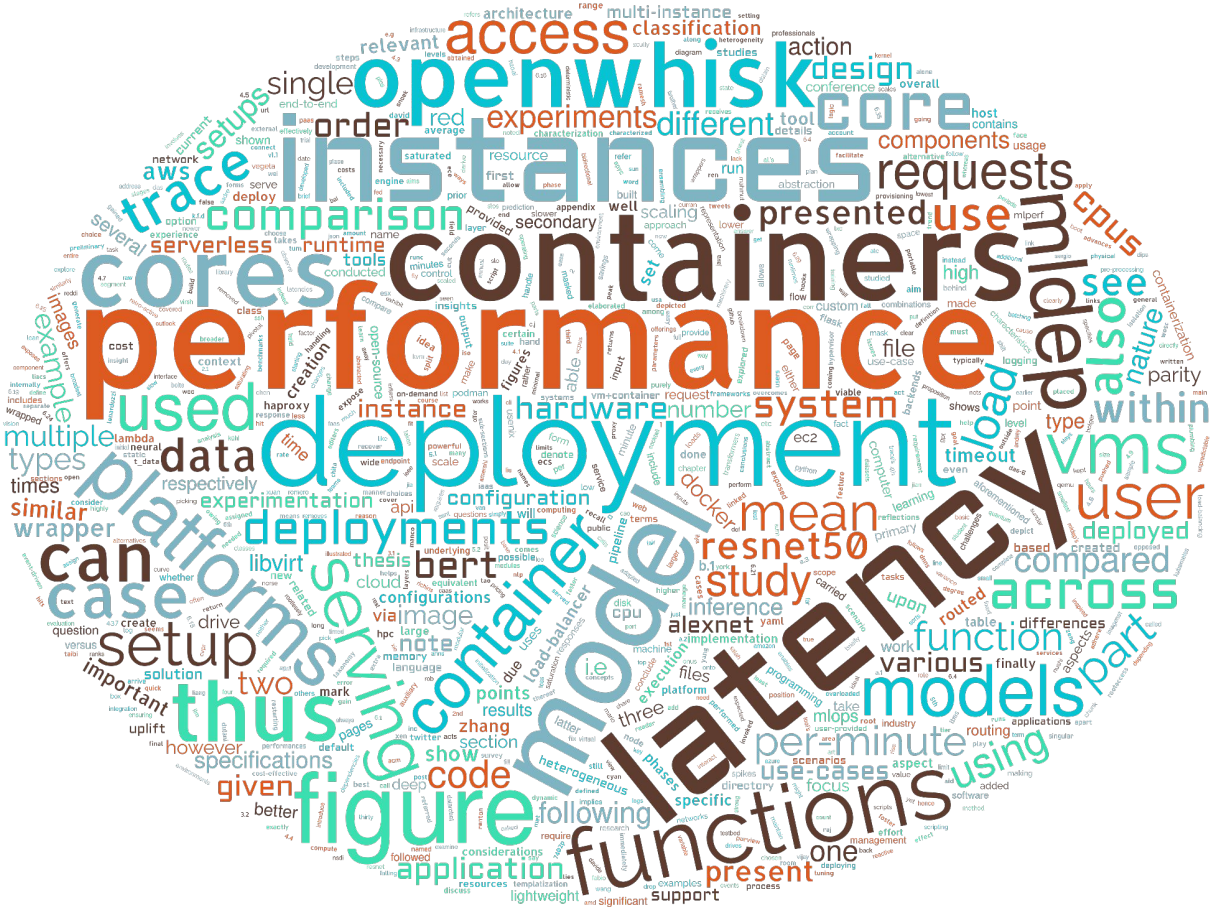


Experiments

- Performance across different deployment platforms and **instance types**
  - Single instance, how does performance scale with **increasing cores**?
- Performance Across **multiple instances of a single deployment platform**
  - Multiple instances with access to 2 cores each, how does performance scale with **increasing instances**?
  - Running on the same physical host, no cores shared
  - Instances vs. cores
- **Containers versus functions** in **conjunction with VMs**
  - (VM+Container) vs. (VM+Function)
  - Heterogeneous vs. Multi-Instance

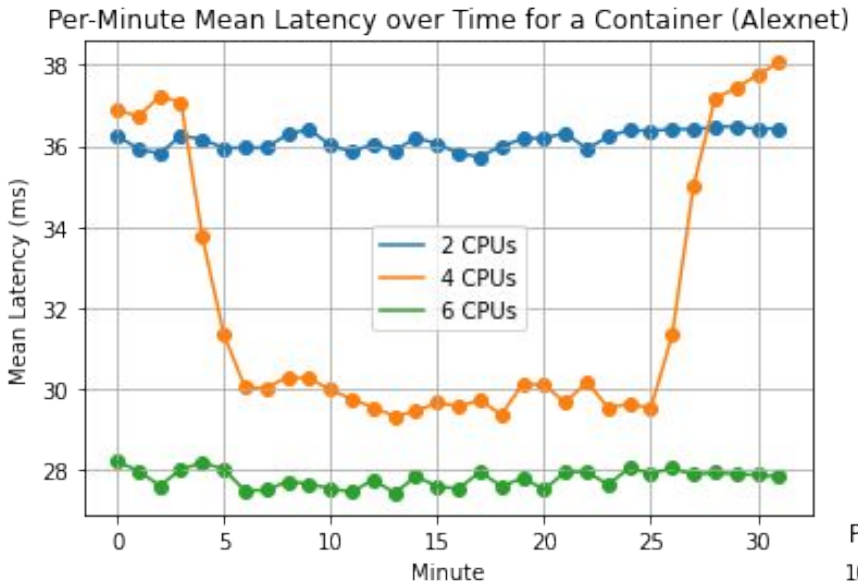
Overview

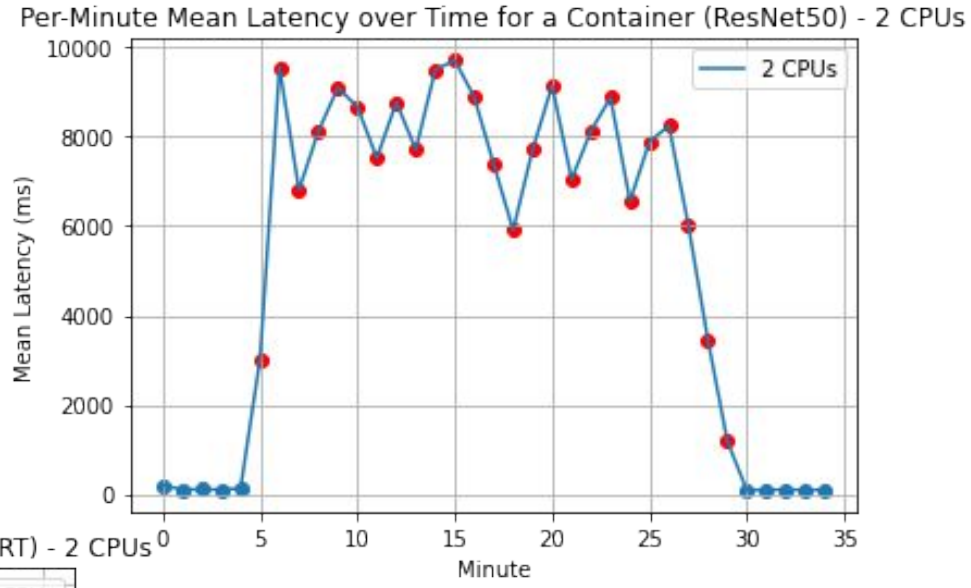
- Introduction
- Research Questions
- MLDep - Design and Implementation
- Experimentation Approach
- **Results**
- Conclusion
- Future Work

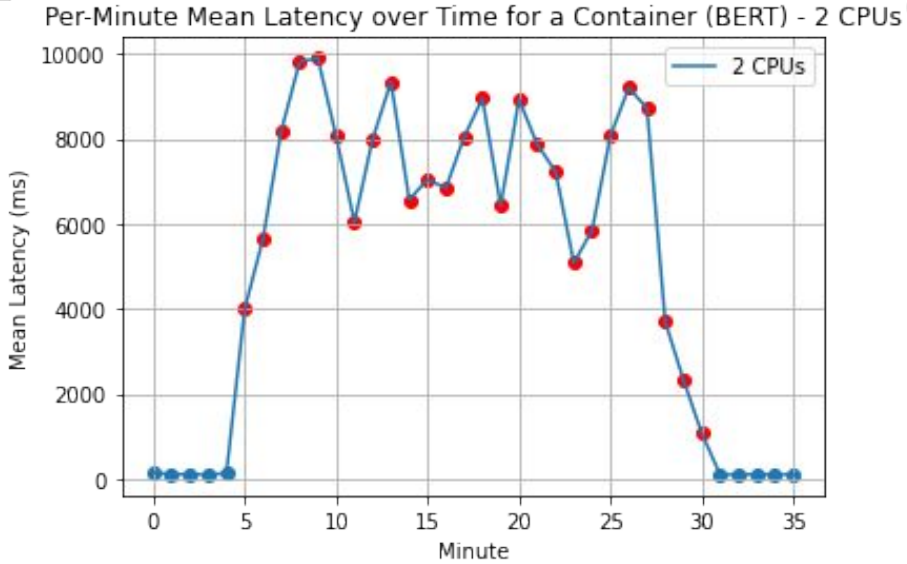

latency
containers
performance
deployment
instances
openwhisk
mldep
model
functions
models
platforms
deployments
cores
used
data
thus
figure
setup
case
can
serverless
trace
comparison
access
requests
vms
user
mean
resnet50
bert
across
compared
function
per-minute
using
part
also
within
see
cpus
core
design
experiments
different
presented
use
system
hardware
instance
configuration
experimentation
code
application
present
one
support
lightweight
specifications
given
following
two
three
image
configurations
alexnet
inference
various
study
deployed
docker
container
load-balancer
runtime
tools
order
single
relevant
multi-instance
classification
architecture
action
components
secondary
scaling
timeout
parity
file
use-cases
mlops
routing
specific
considerations
resources
templatization
deploying
tuning
process
examples
significant
lightweight
multiple
types
times
similar
wrapper
libvirt
thesis
cloud
via
note
section
zhang
results
points
heterogeneous
phases
execution
implementation
programming
aspects
differences
routed
upon
carried
work
finally
applications
software
added
important
however
uplift
question
drive
versus
solution
better
aws
several
setups
images
example
creation
haproxy
response
lambda
neural
static
t_data
open-source
cost
context
experience
option
network
shown
connect
deploy
serve
vgpu
vt-x
data
resource
saturated
characterization
characteristics
average
first
prior
engine
aims
provided
slower
appendix
well
built
scaling
approach
lower
provisioning
mlperf
abstract
usage
host
contains
overall
conference
studies
professionals
heterogeneity
diagram
along
steps
levels
estimated
development
infrastructure
range
end-to-end
current
external
effectively
shown
connect

Scaling Across Cores

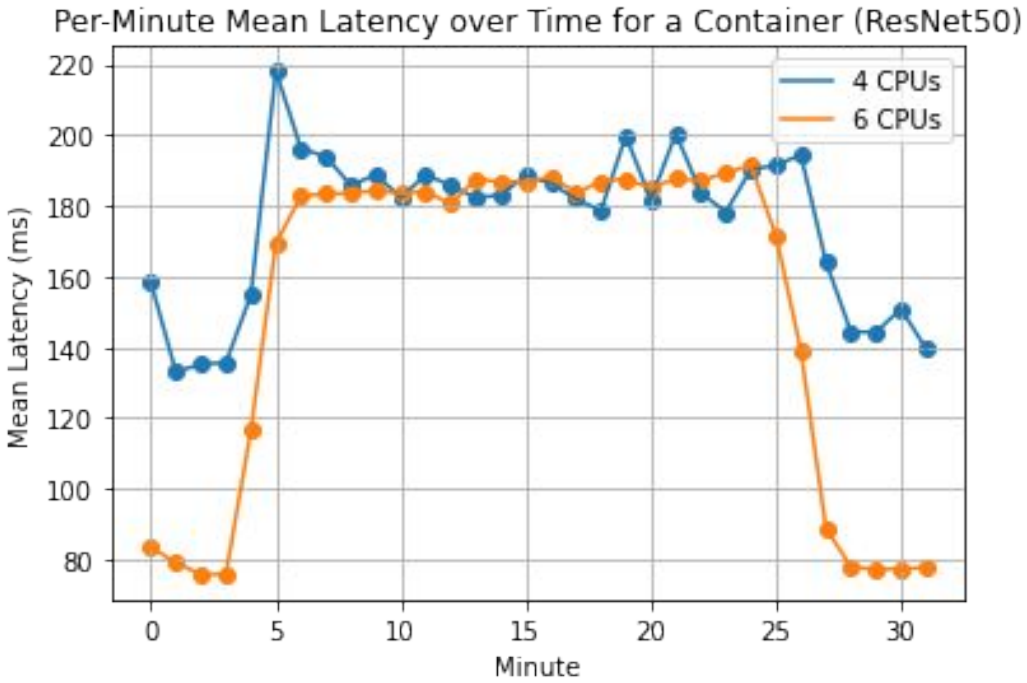
Containers - Scaling Across Cores

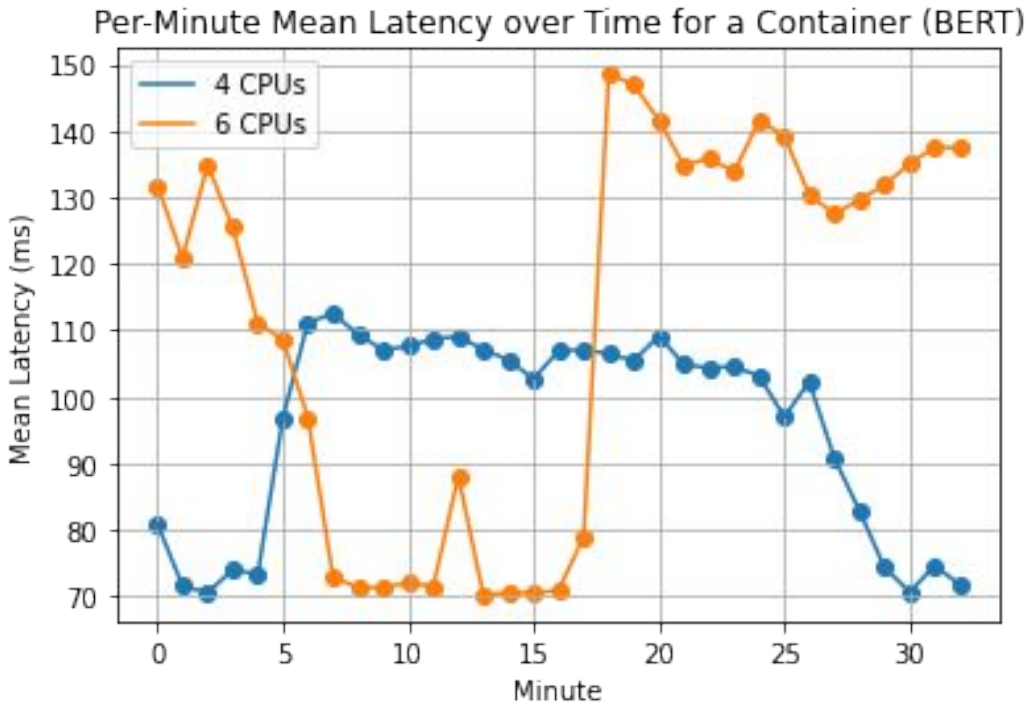






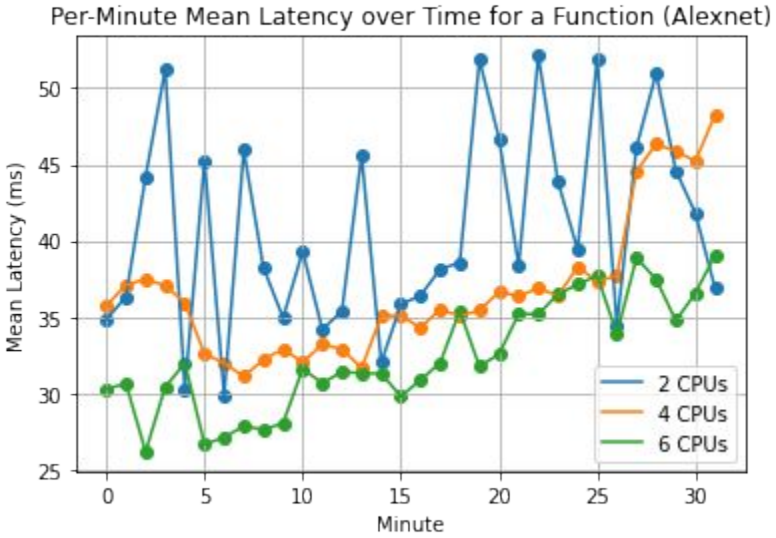
Containers - Scaling Across Cores

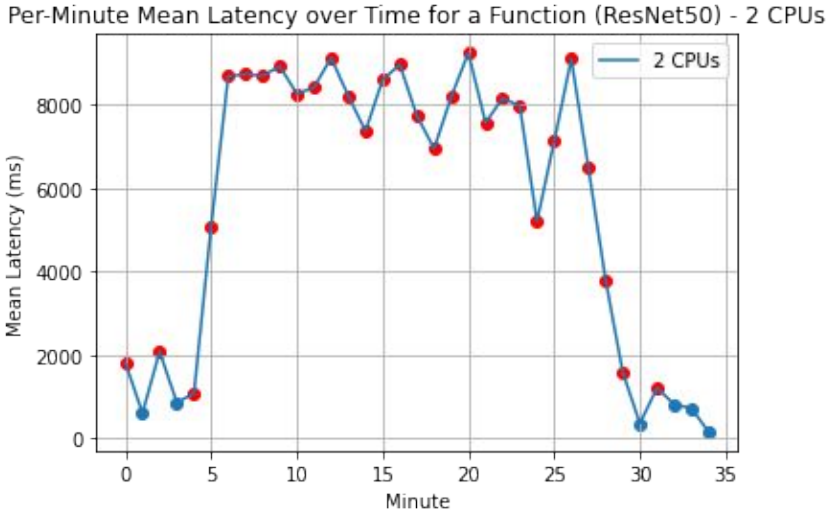


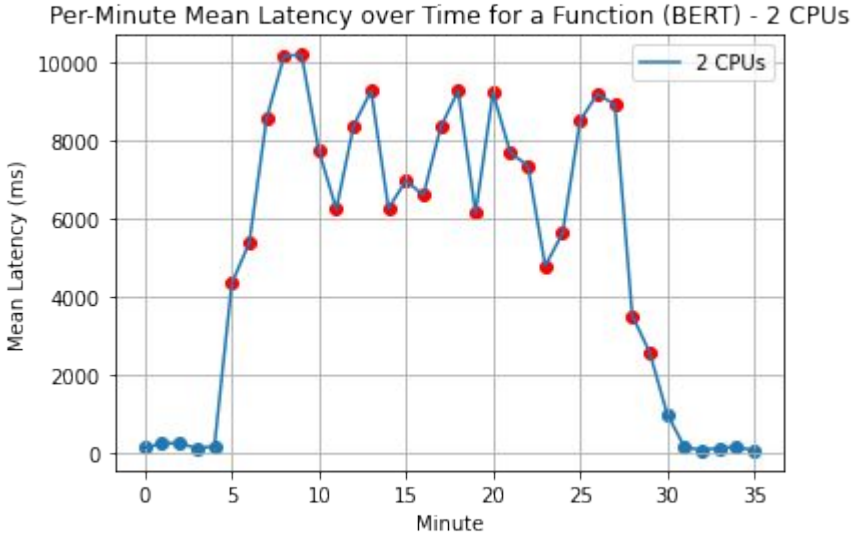


- With 2 cores we do see containers recover in the final third of the trace
- Once saturation is overcome, adding more cores leads to diminishing or no returns
- Averages include timeouts! Without this inclusion, 2 cores do in fact degrade gracefully compared to 4 and 6 cores

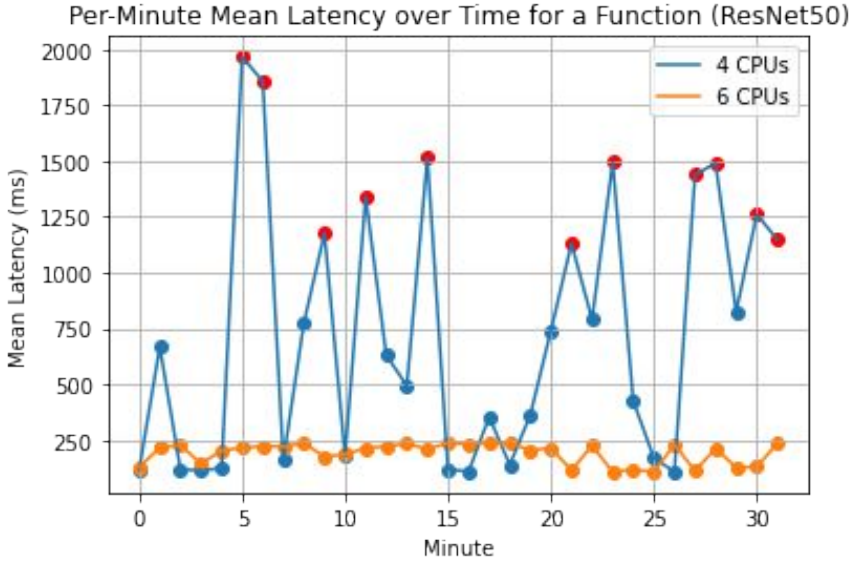
Functions - Scaling Across Cores

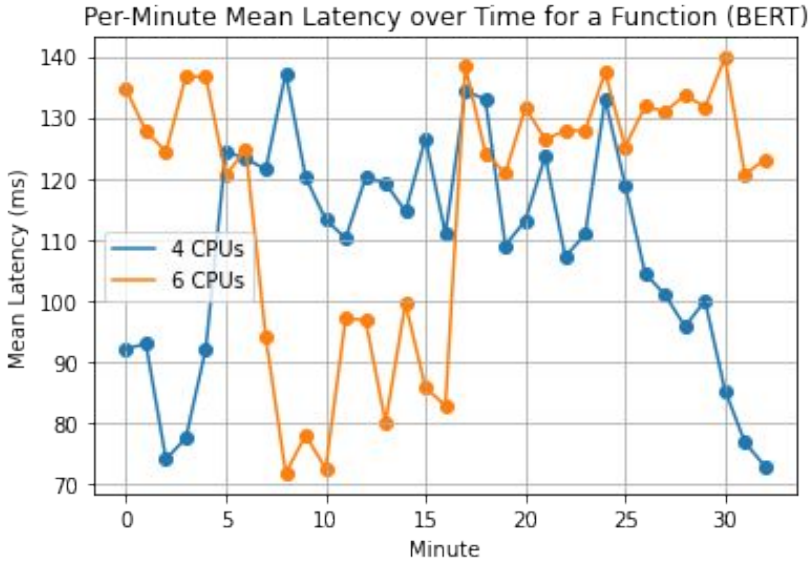




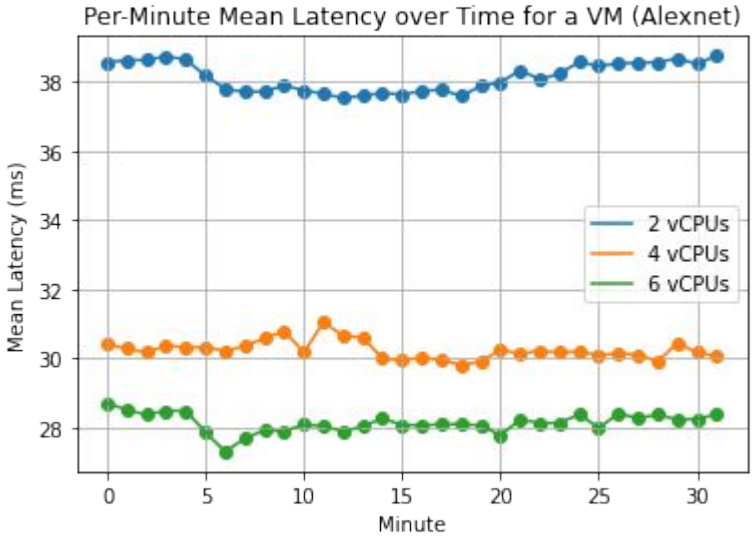


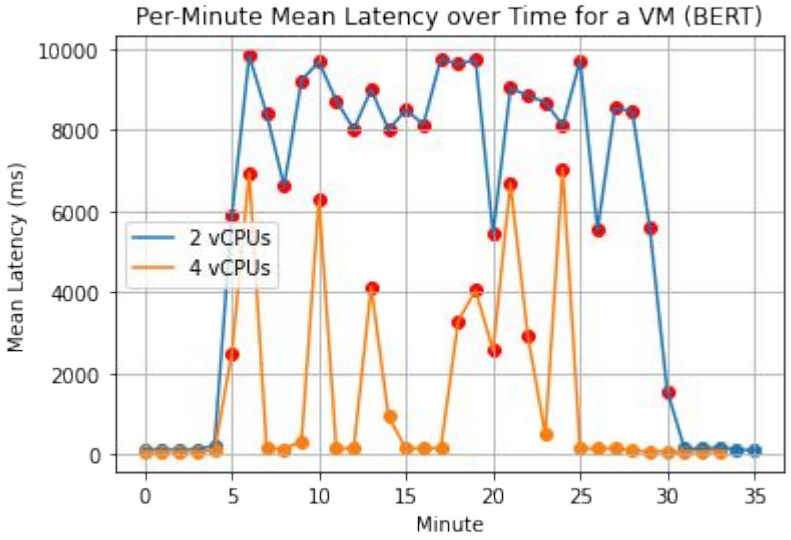
- Functions require more resources than containers
- AlexNet scales to an extent
- Response times more erratic than containers
- Scaling up - inconclusive

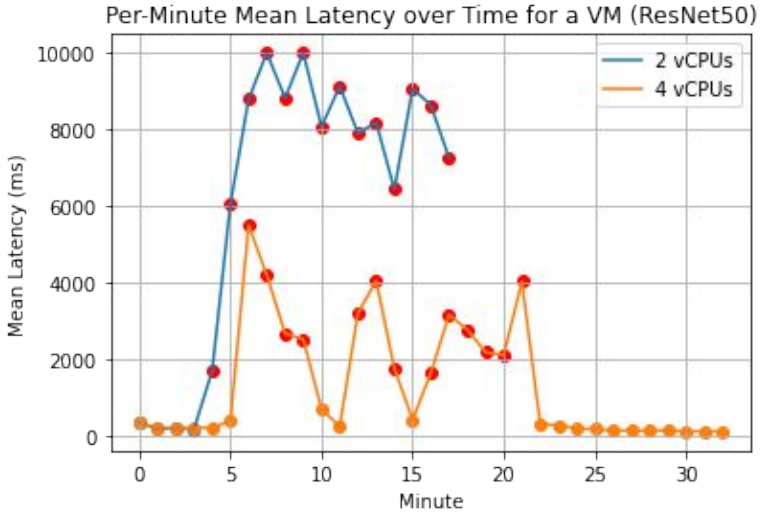




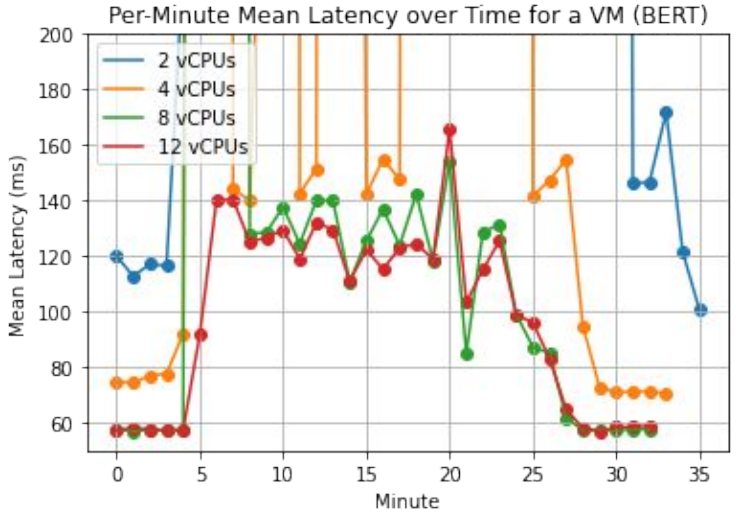
VMs - Scaling Across Cores

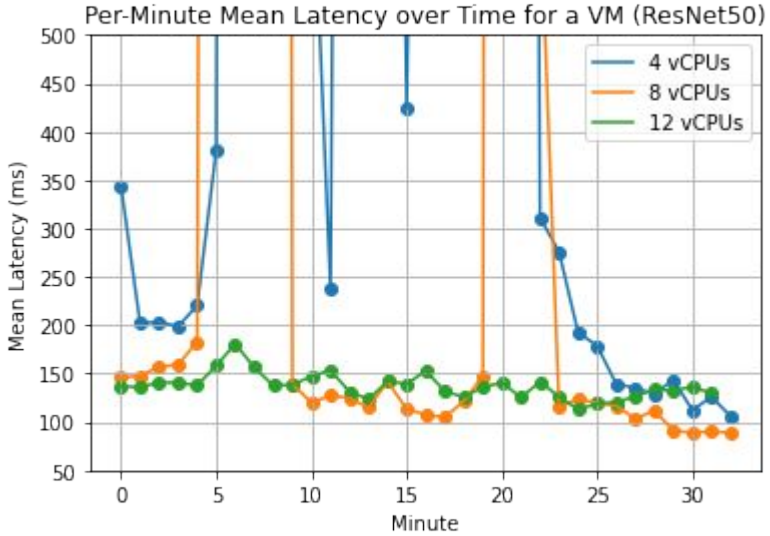






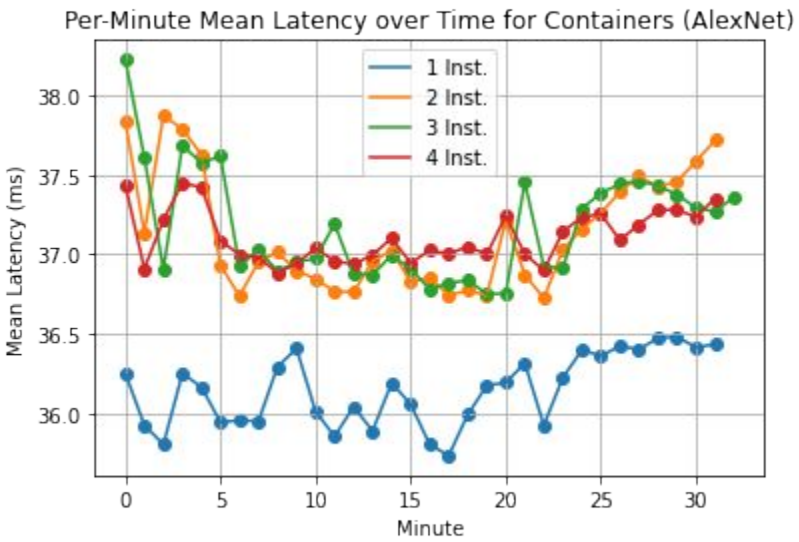
- AlexNet behaves similarly across the 3 platforms
- Even 8 cores seem to get overloaded! Quantum of overloading drops
- Performance uplift inconclusive

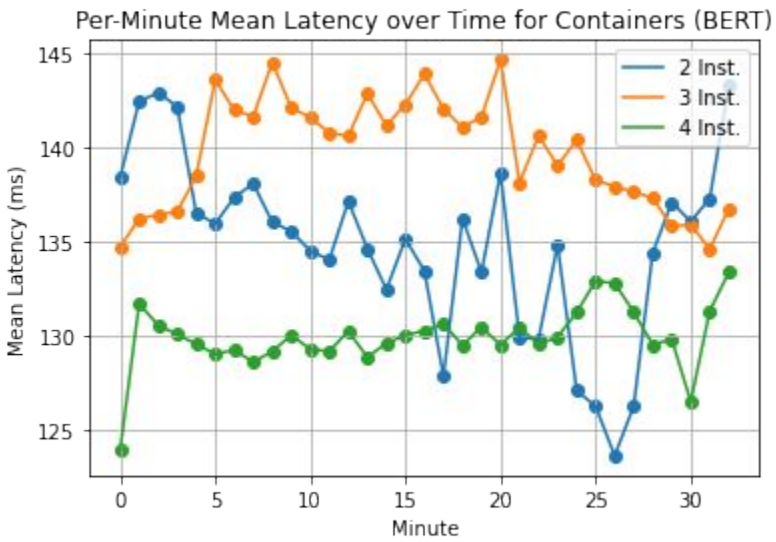


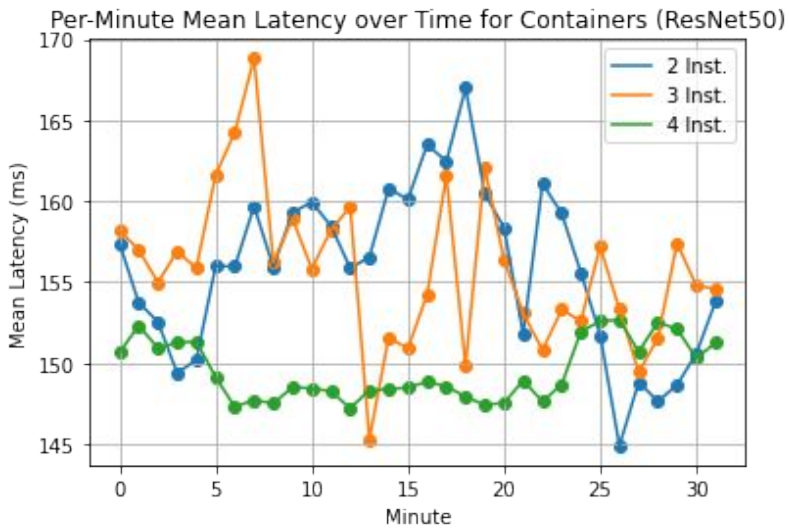


Scaling Across Instances

Containers - Scaling Across Instances

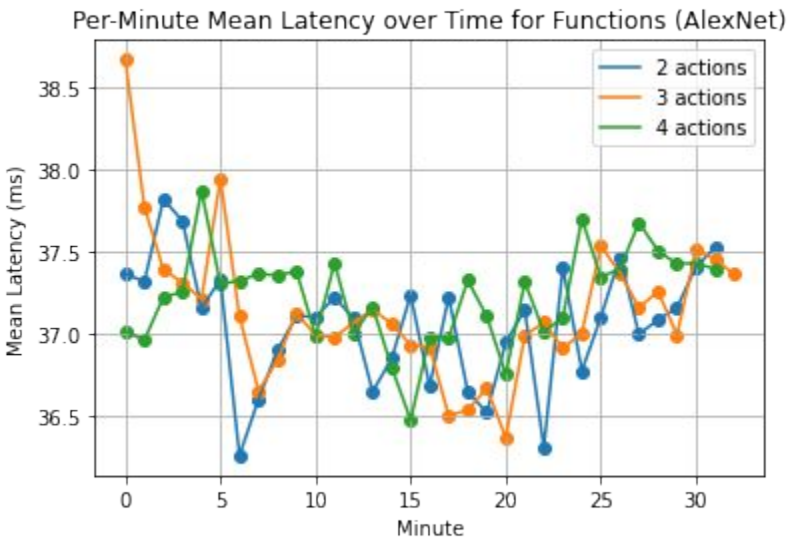


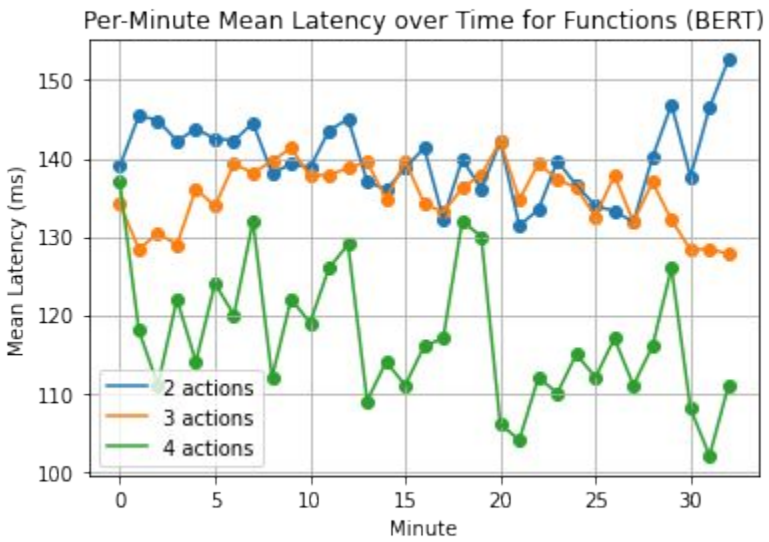


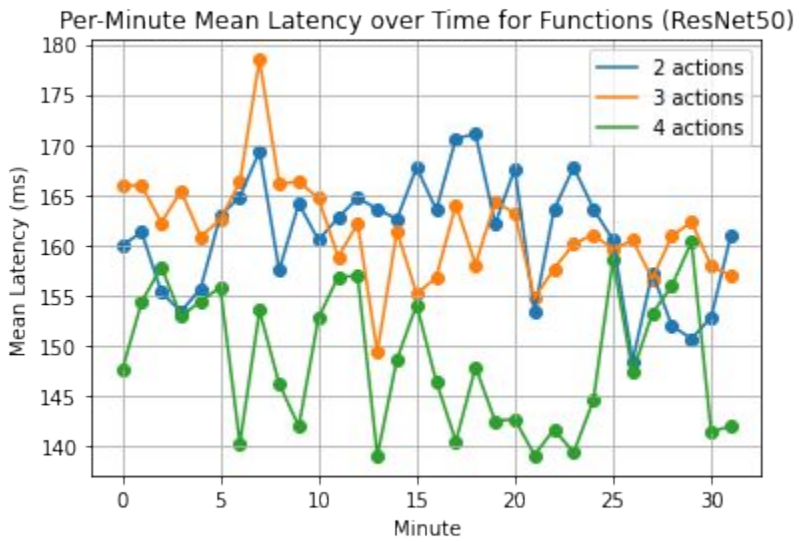


- BERT and ResNet - 4 instances the clear winner, 2 and 3 trade blows
- AlexNet is entirely unintuitive
  - Change in mean latency

Functions - Scaling Across Instances

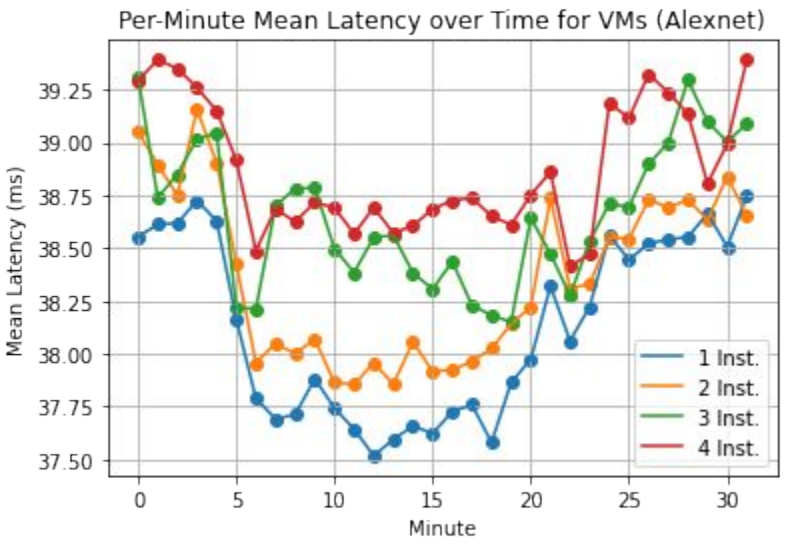


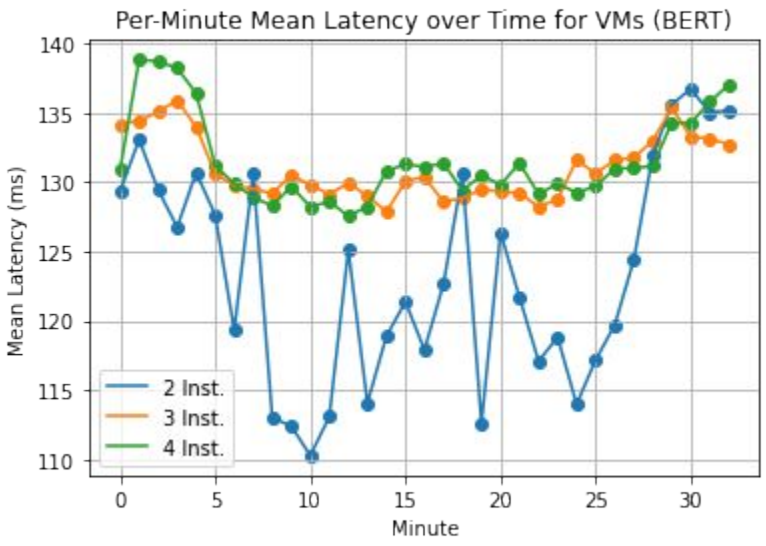


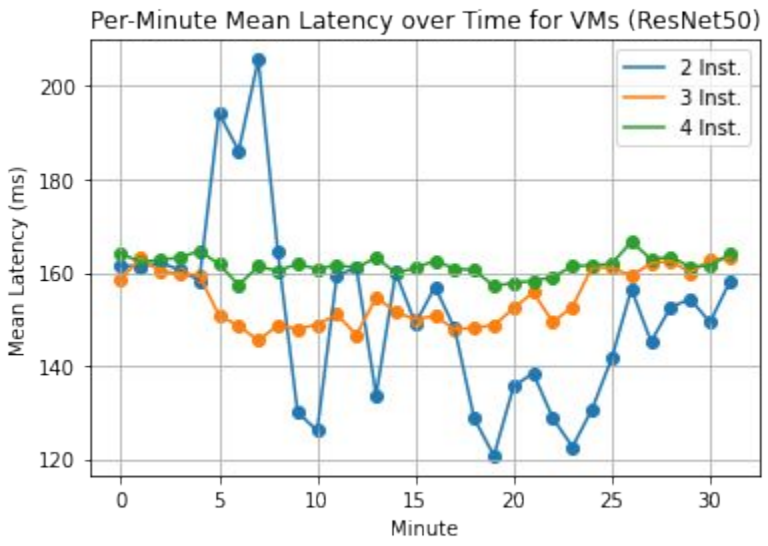


- What do multiple function instances mean? - Multiple containers vs. multiple actions
- With ResNet and BERT, 4 stands apart, 2 and 3 trade blows
- AlexNet does not scale out (Remember containers!)

VMs - Scaling Across Instances







- The setup with minimum number of instances that can effectively handle peak load wins
- 3 and 4 instance setups yield more predictable latencies
- 4 instances of 2 cores each flattens the curve better than a single 8 core instance

Scaling - Cores vs. Number of Instances

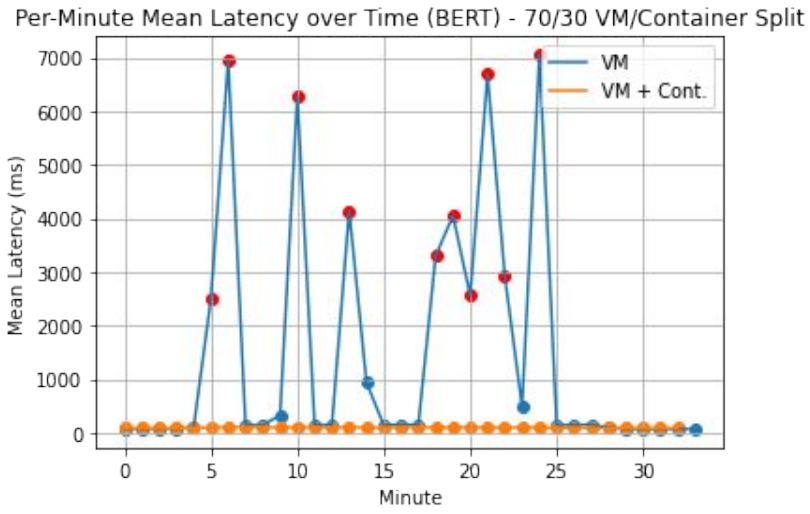
	Setup	Mean Latency (ms)	Setup	Mean Latency (ms)
Container	4 cores	175.7	6 cores	151.3
	2 instances	155.9	3 instances	155.8
Function	4 cores	Overloaded	6 cores	191.7
	2 instances	161.2	3 instances	160.3
VM	4 cores	Overloaded	8 cores	Overloaded
	2 instances	151.3	4 instances	161.5

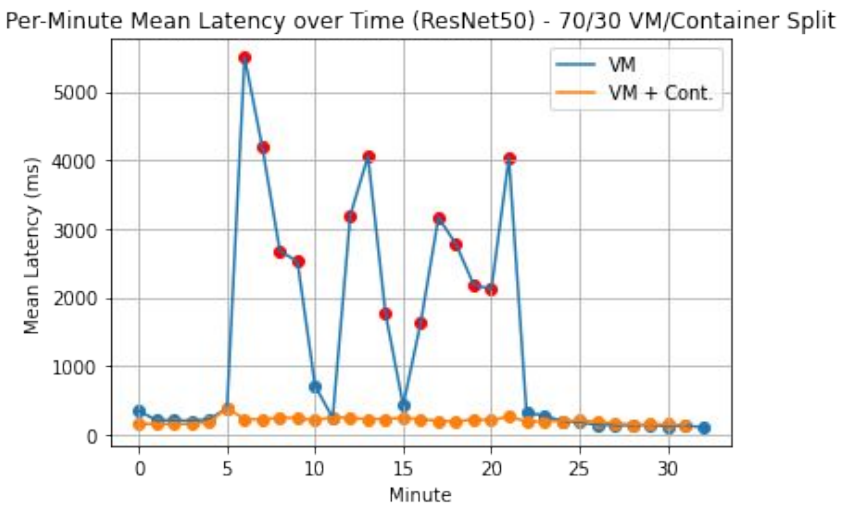
- Learning -
  In the case of equivalent hardware, scaling out less powerful hardware > scaling up
- Caveat - hardware parity, cost benefit
- Increasing number of instances is a good option, but is there another?

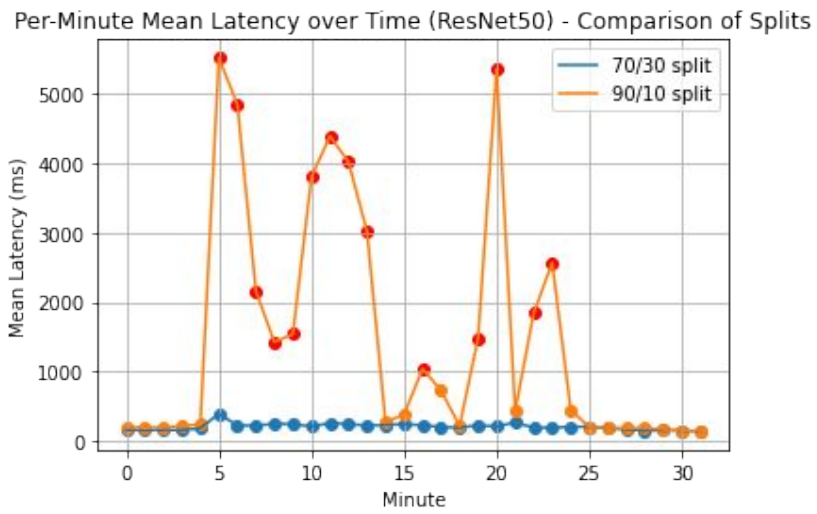
c5.large	$0.062	27%	$0.085
c5.xlarge	$0.123	28%	$0.17
c5.2xlarge	$0.246	28%	$0.34
c5.4xlarge	$0.493	28%	$0.68

Heterogeneous Setups

Heterogeneous Setups - Containers as the Secondary Deployment

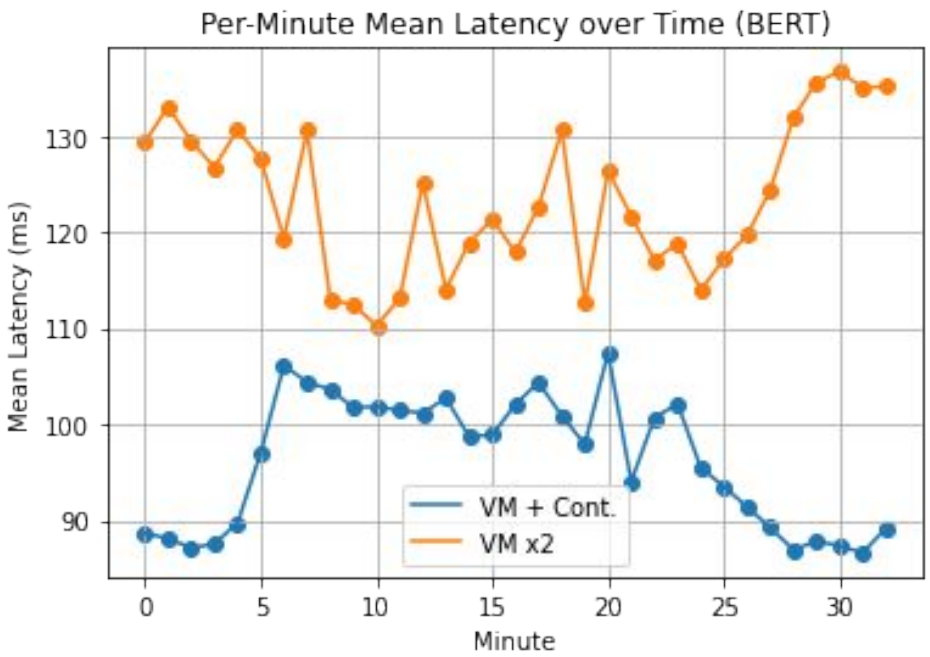


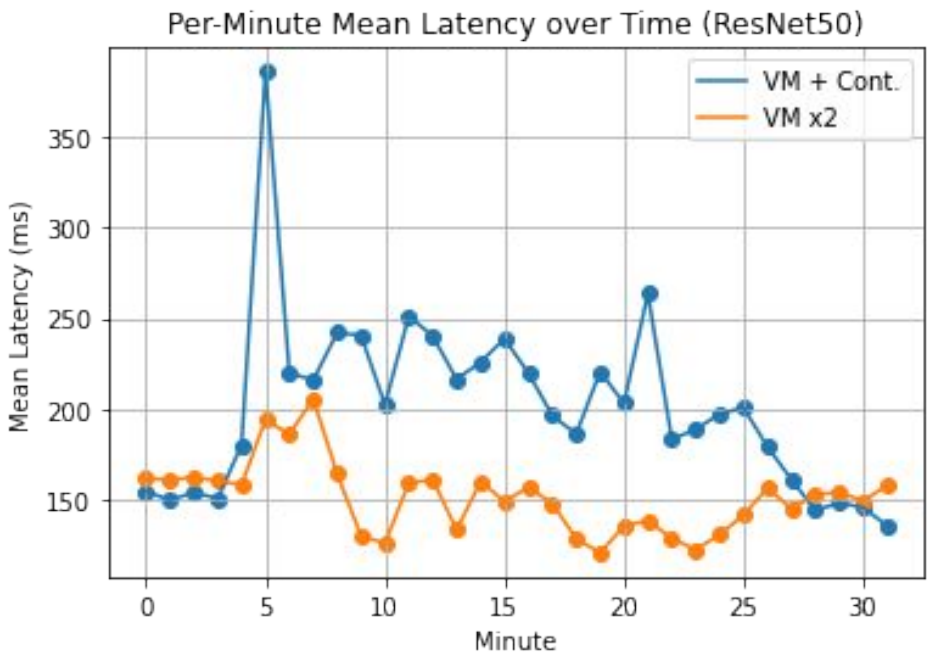




- The secondary deployment helps overcome saturation
  - Push the primary as much as possible, routing
- A 6 core VM would show inferior performance!
- What about comparison against pure VMs or container setups?

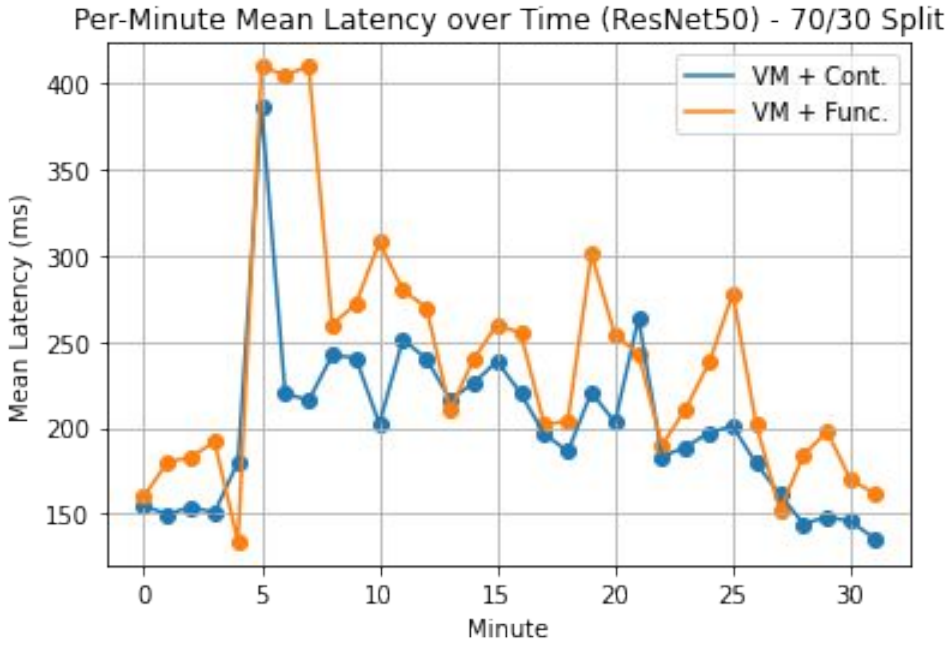
Heterogeneous Setups vs Homogeneous Setups





- A heterogenous setup can compete
- Similar results when compared against a container with 4 cores
- Comes down to cost, further experimentation

Heterogeneous Setups - Containers vs Functions as the Secondary Deployment



- Results are intuitive - container > function
- Cost - Huge caveat! Functions are PAYG. Is the performance uplift of containers worth it?

Discussion

Discussion * Terms and Conditions Apply

- Hardware Parity - not the case with cloud platforms
- Containers, Functions, VMs OR
  Docker, OpenWhisk, KVM
- Costs, pricing models

Discussion

- At parity - VMs require the most resources to drive equivalent performance, Functions require lesser, Containers the least

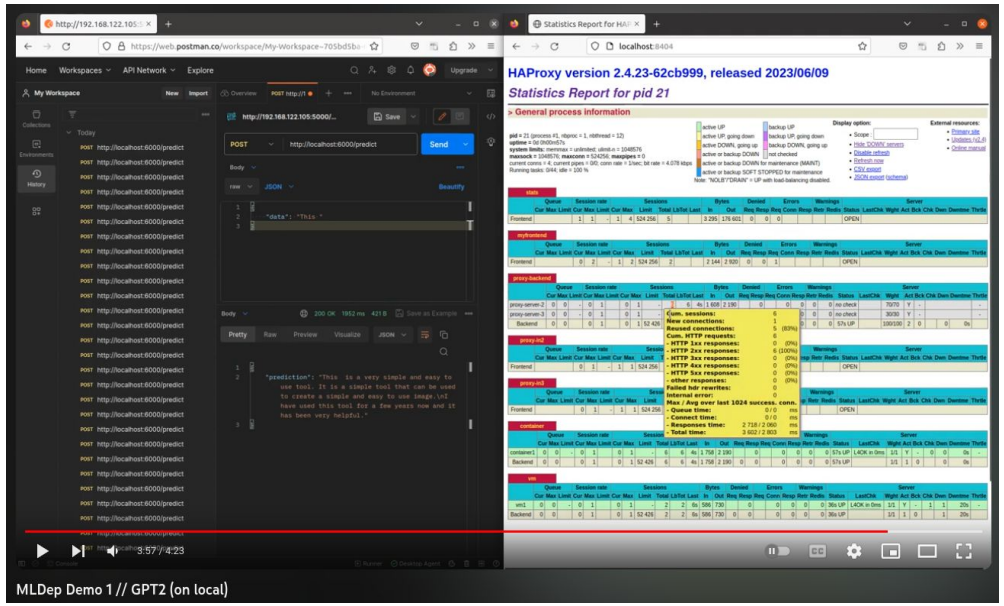
- Settle on a core count that does not get overloaded or saturated
  - Gains are inconclusive beyond this

- For all three platforms, scaling out to multiple instances is highly effective
  - Scaling out beyond a point nets inconclusive gains
  - Flattens the response times, **higher predictability**

- Alternative to multi-instance, route a small fraction of the load to a **secondary deployment**
  - Might not always be better than homogenous setups, but could be more cost-effective
  - Containers act as a better secondary deployment than functions, but pricing model should be considered

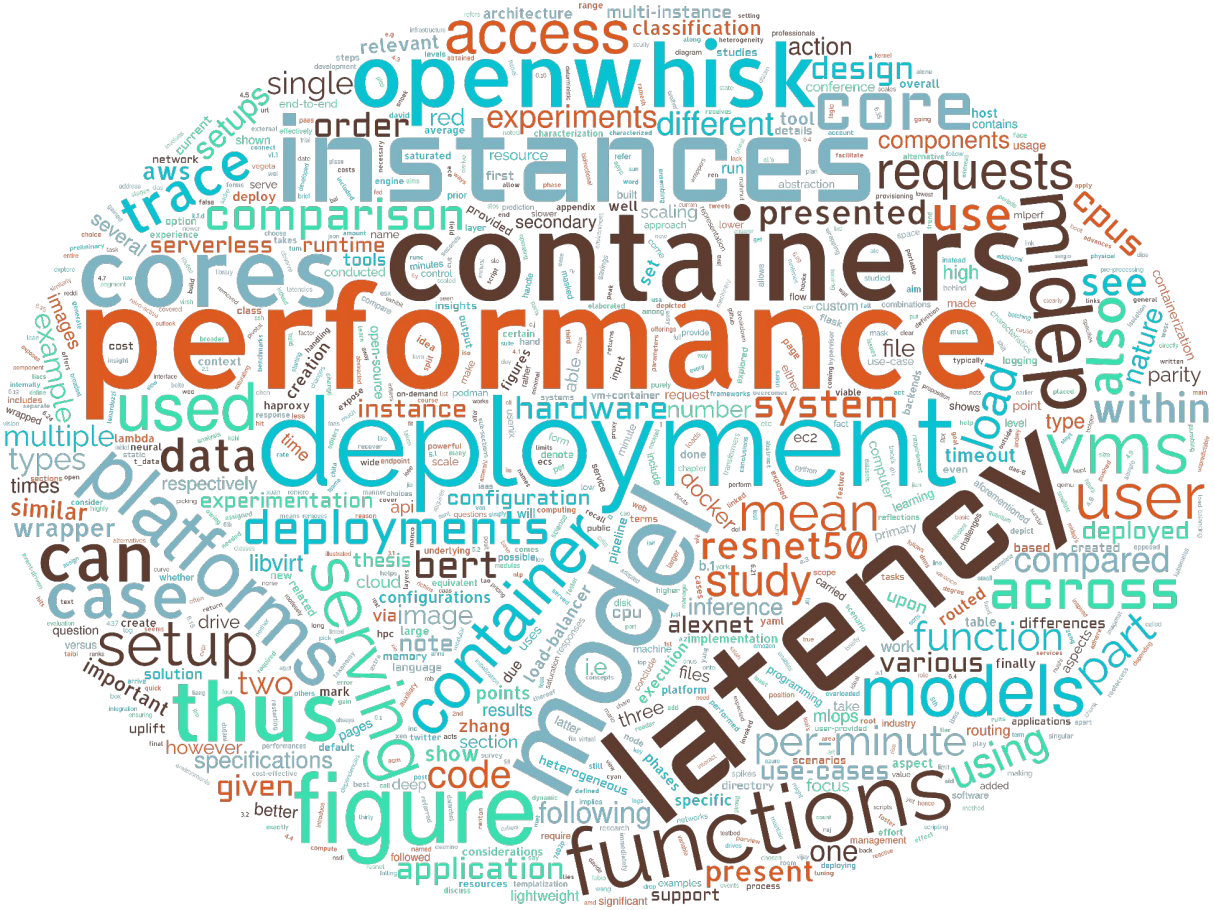
Discussion - What did we miss out on?

- Complete measurement of baseline performances and a no-op measurement
- Wider range of models, heavier models
- Wider range of hardware setups, instance numbers
- Further investigation into performance discrepancies
  - Containers vs. Functions
  - That 4 core AlexNet container!



Overview

- Introduction
- Research Questions
- MLDep - Design and Implementation
- Experimentation Approach
- Results
- **Conclusion**
- Future Work



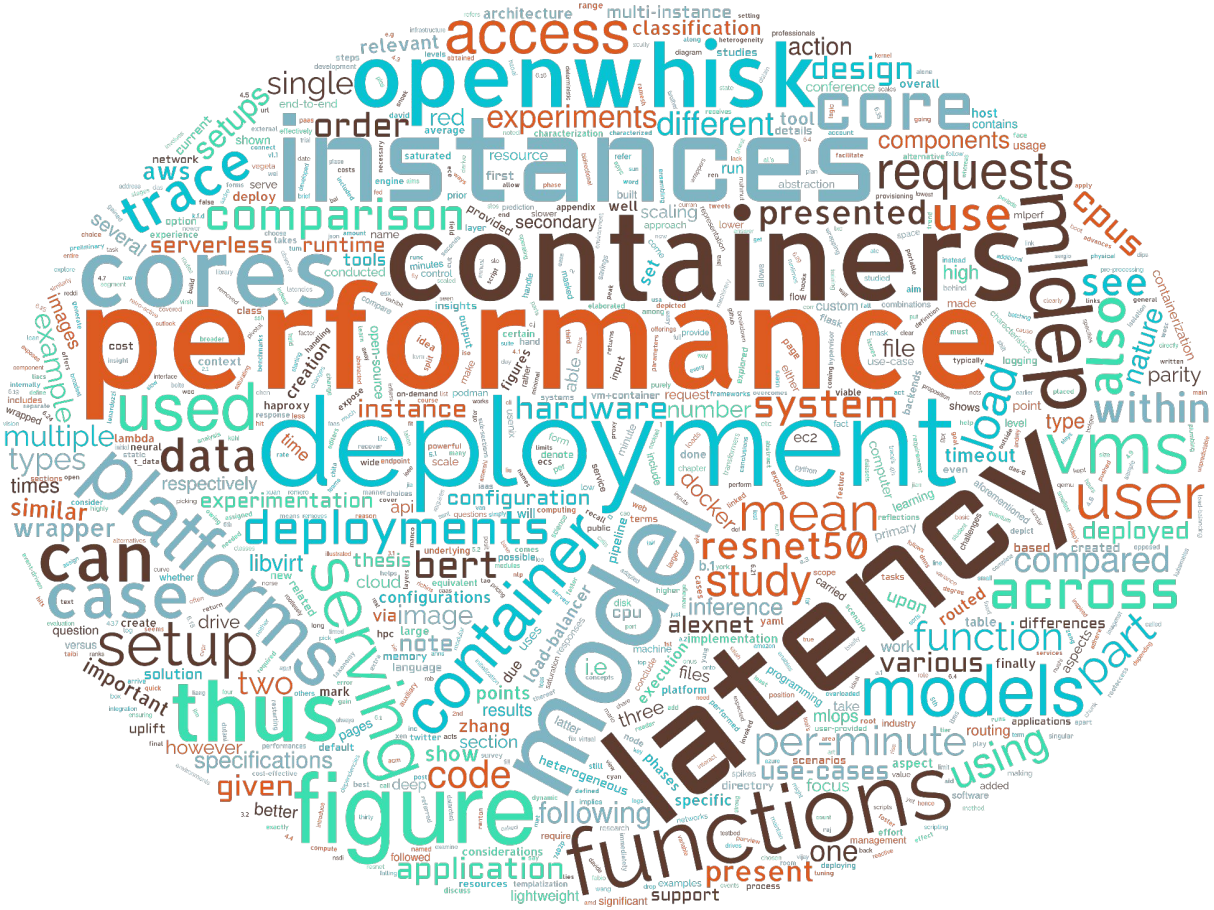
Conclusion

- Potential challenges and pitfalls, comparative performance
  - VMs require more powerful resources to drive performance
  - Functions show less predictable response times than containers
  - Scaling up and out is very contextual, not always predictable

- Multi-instance and heterogeneous setups are good solutions that each have their own caveats

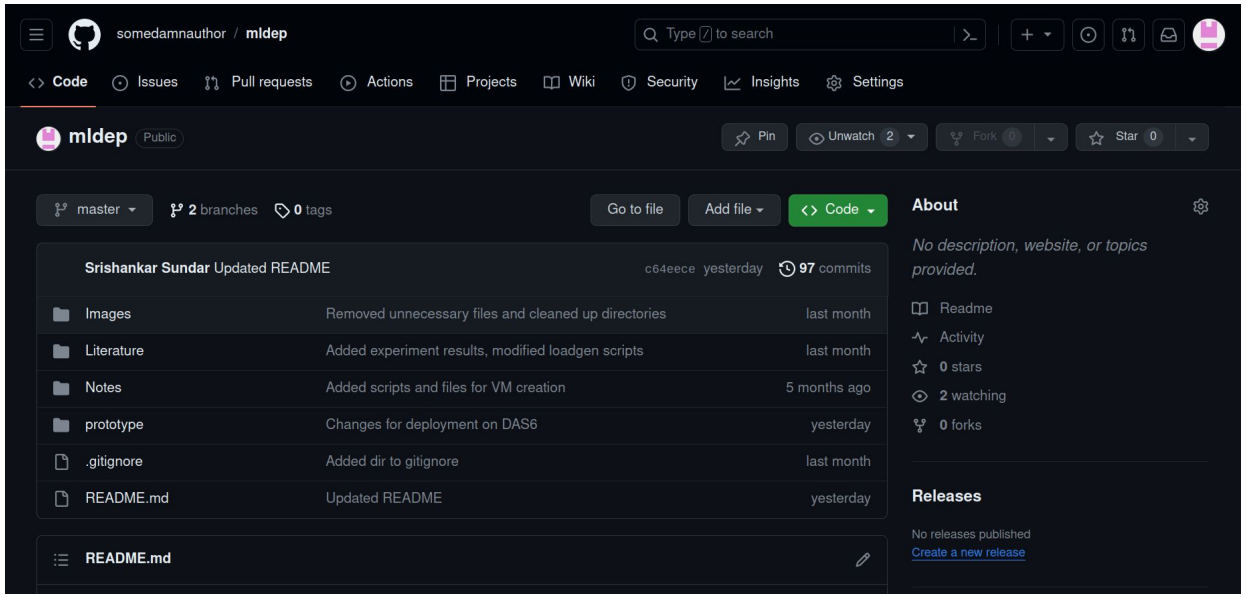
Overview

- Introduction
- Research Questions
- MLDep - Design and Implementation
- Experimentation Approach
- Results
- Conclusion
- **Future Work**


latency
performance
deployment
containers
instances
openwhisk
mldep
model
models
functions
figure
thus
platforms
serving
cores
used
case
can
setup
data
deployments
container
access
core
requests
vms
user
mean
resnet50
study
across
compared
function
part
using
per-minute
use-cases
code
application
present
one
support
following
given
specifications
experiments
different
comparison
trace
serverless
runtime
tools
aws
setups
order
single
relevant
classification
multi-instance
architecture
action
design
components
presented
use
cpus
see
also
within
nature
containerization
parity
type
timeout
deployed
load
system
hardware
number
instance
configuration
experimentation
respectively
times
similar
wrapper
multiple
types
images
example
several
secondary
scaling
well
built
resource
saturated
first
run
abstraction
provided
slower
appendix
set
control
conducted
open-source
creation
haproxy
docker
computer
learning
reflections
primary
terms
pipeline
public
machine
cpu
disk
alexnet
inference
implementation
work
various
routing
applications
differences
aspects
finally
routed
upon
carried
tasks
created
based
appeared
mlperf
provisioning
knowledge
alone
overall
conference
host
contains
usage
tool
details
characterization
account
facilitate
alternative
following
apply
physical
pre-processing
general
containerization
nature
written
main
placed
made
high
point
level
shows
file
use-case
backends
viable
request
frameworks
vm+container
ec2
lambda
neural
static
t_data
includes
wrapped
indirectly
explored
exposed
either
page
cost
context
hand
insight
idea
figure
output
making
able
input
time
minute
delay
service
scale
denote
ecs
endpoint
wide
done
perform
typically
logging
custom
flask
mlops
take
aspect
scenarios
user-provided
industry
routing
applications
apart
singular
software
added
method
scripting
effort
management
deploying
tuning
active
back
process
examples
discuss
templatization
resources
lightweight
significant
considerations
followed
deep
best
cost-effective
performances
twitter
acts
section
show
zhang
heterogeneous
phases
specific
directory
focus
scenarios
aspect
routing
mlops
take
work
implementation
programming
platform
execution
machine
memory
language
note
image
cloud
configurations
thesis
bert
libvirt
related
new
whether
drive
question
versus
text
create
load-balancer
uses
due
points
results
three
however
pages
default
mark
solution
important
uplift
better
deep
latter

Future Work

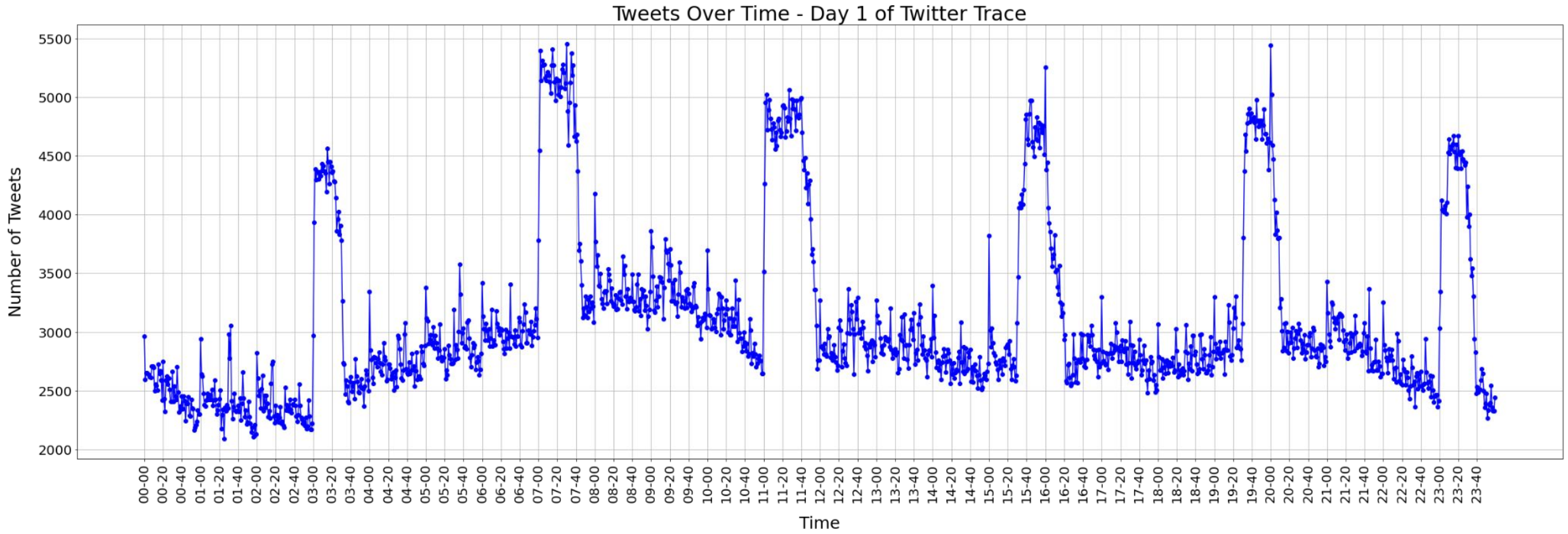
- MLDep as a testbench
  - Maturity in terms of design
  - Extension to cloud offerings
  - Reactive scaling
  - Support for more solutions



Thank You!

Universiteit Leiden
The Netherlands

Appendix


Tweets Over Time - Day 1 of Twitter Trace
Number of Tweets
5500
5000
4500
4000
3500
3000
2500
2000
Time
00-00
00-20
00-40
01-00
01-20
01-40
02-00
02-20
02-40
03-00
03-20
03-40
04-00
04-20
04-40
05-00
05-20
05-40
06-00
06-20
06-40
07-00
07-20
07-40
08-00
08-20
08-40
09-00
09-20
09-40
10-00
10-20
10-40
11-00
11-20
11-40
12-00
12-20
12-40
13-00
13-20
13-40
14-00
14-20
14-40
15-00
15-20
15-40
16-00
16-20
16-40
17-00
17-20
17-40
18-00
18-20
18-40
19-00
19-20
19-40
20-00
20-20
20-40
21-00
21-20
21-40
22-00
22-20
22-40
23-00
23-20
23-40