STAROID

staroid.com

How we built Spark serverless on top of Kubernetes

Moon soo Lee / moon@staroid.com

whoami

Moon soo Lee (moon@staroid.com)

Creator of Apache Zeppelin
Founder at Staroid

Staroid

Deliver open source software as a service and fund developers

https://staroid.com

STAROID

Traditional Spark cluster

User

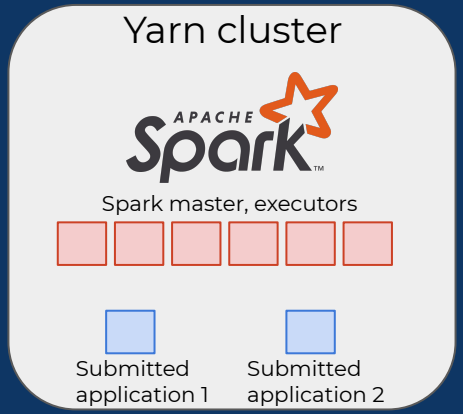
Configure network (VPC, subnet)

Deploy a cluster n x master, m x worker node

Get network access to Spark cluster to submit job (SSH to node or connect to notebook running in the same network)

Package your application

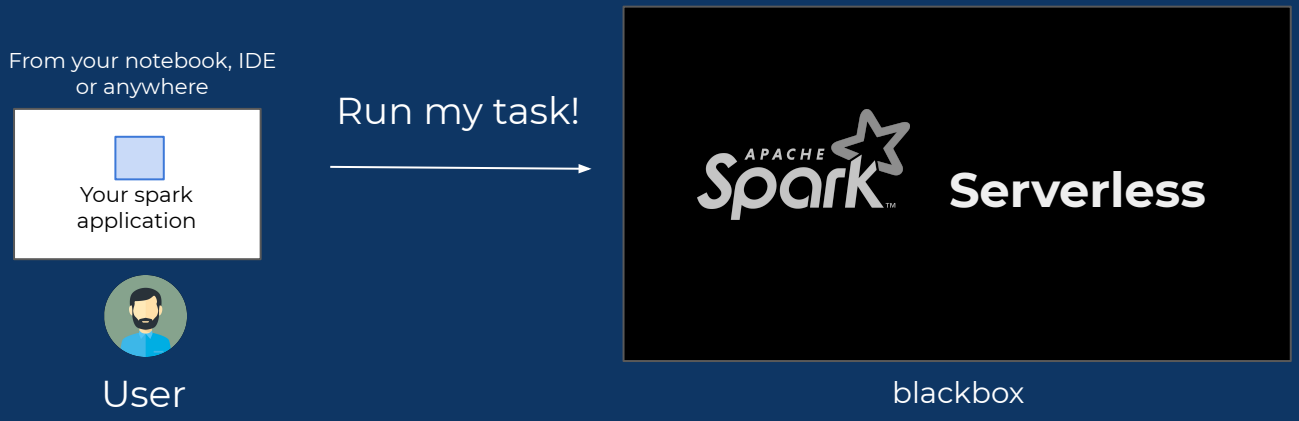
Submit the application



STAROID

Spark serverless

- User run Spark application from any environment (notebook, IDE, etc)
- No app packaging, No job submit, No network configuration



STAROID

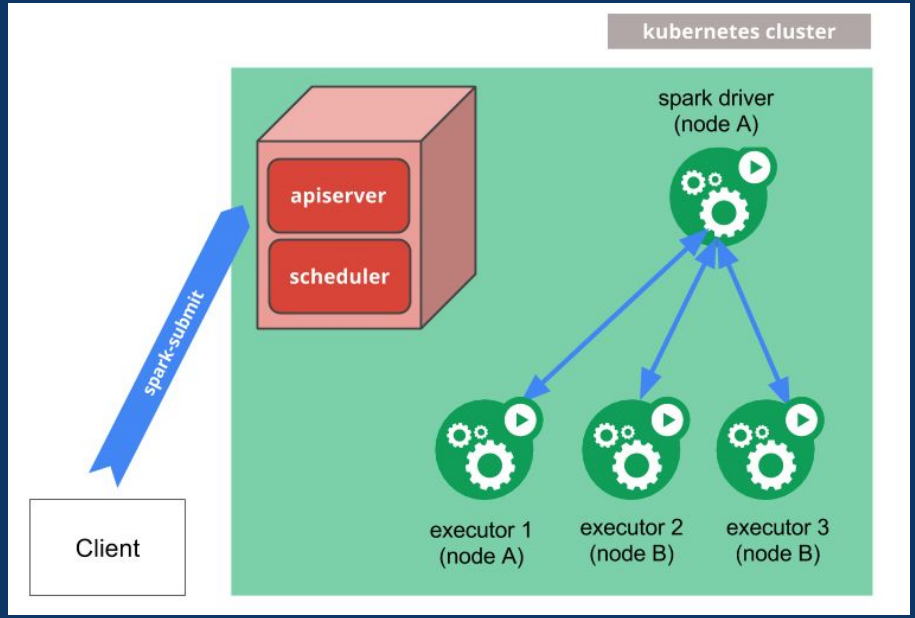
How it works

Let’s first see demo

STAROID

Spark on Kubernetes

- Spark driver and executors are running as a Pod
- Executors can be created dynamically
- When driver Pod finishes, executors are automatically cleaned up
- Means each driver can dynamically create its own containerized executor set



Spark on Kubernetes provides on-demand spark cluster! Let's use it to build spark-serverless!

STAROID

Spark on Kubernetes

You need 4 things to run Spark application in cluster mode

1. Access to Kubernetes API server
   with Rbac permission to create/delete Pod

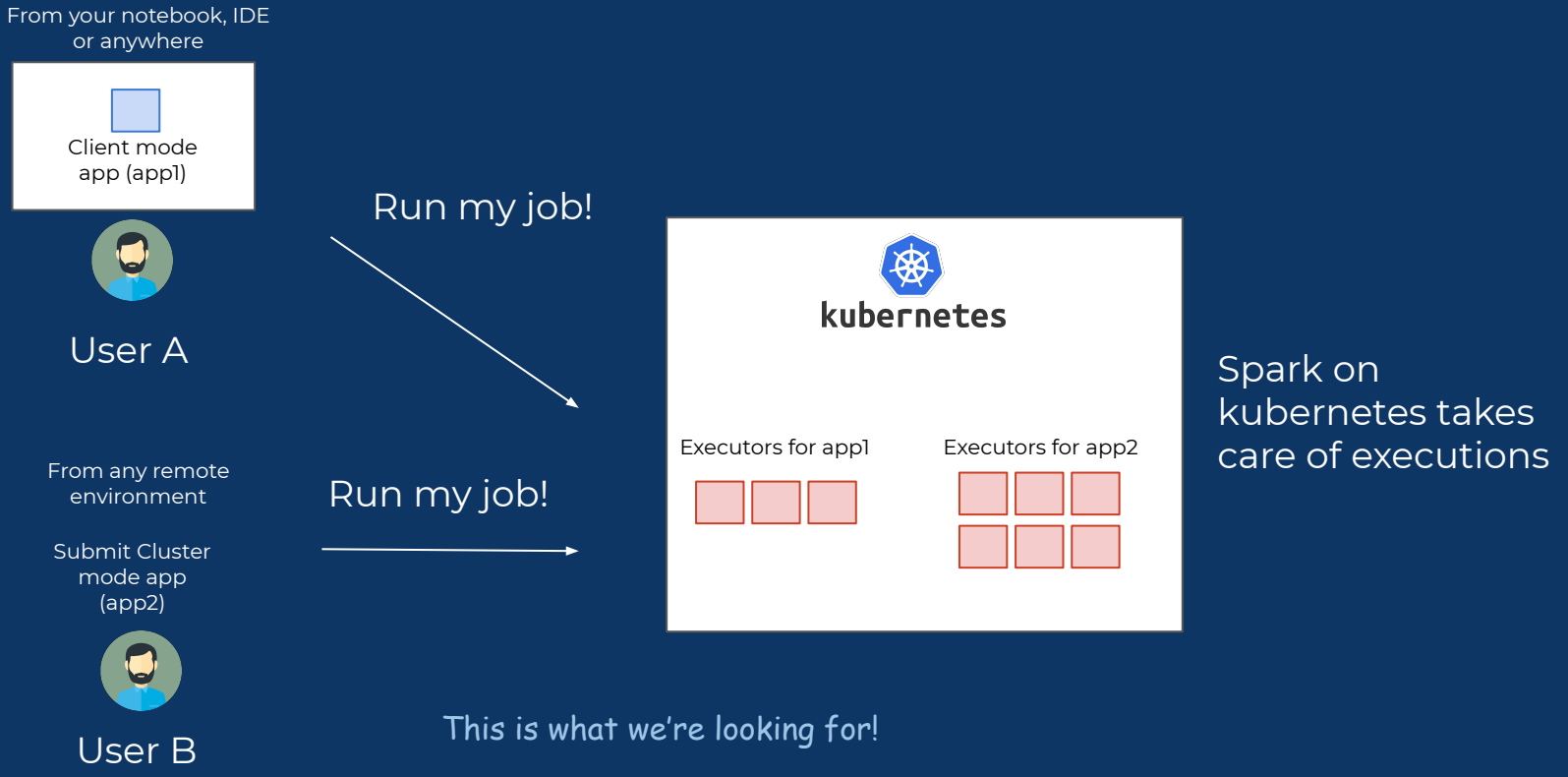
```
$ ./bin/spark-submit \
    --master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
    --deploy-mode cluster \
    --name spark-pi \
    --class org.apache.spark.examples.SparkPi \
    --conf spark.executor.instances=5 \
    --conf spark.kubernetes.container.image=<spark-image> \
     local:///path/to/examples.jar
```

3. Your application artifact

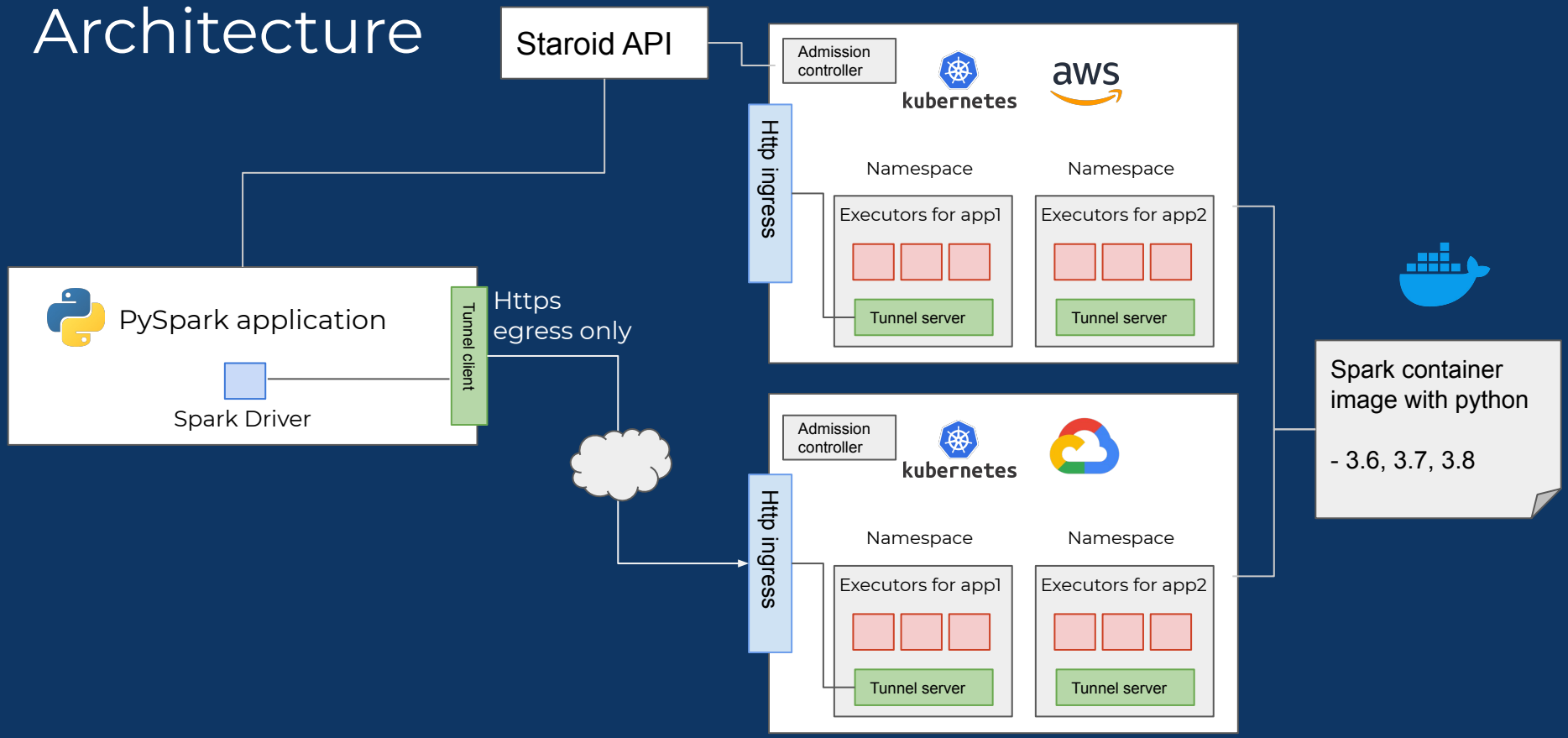
2. Spark container image

4. Network access between Driver Pod and Executor Pods

Spark on Kubernetes



Architecture



Problem 1 - Isolation, Multi-tenancy

From your notebook, IDE or anywhere

Client mode app (app1)

User A

Run my job!

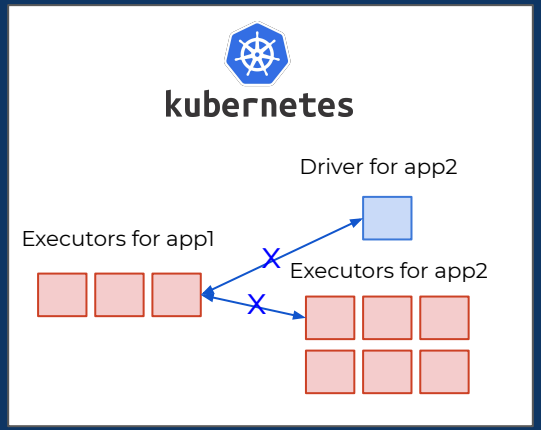
From any remote environment

Submit Cluster mode app (app2)

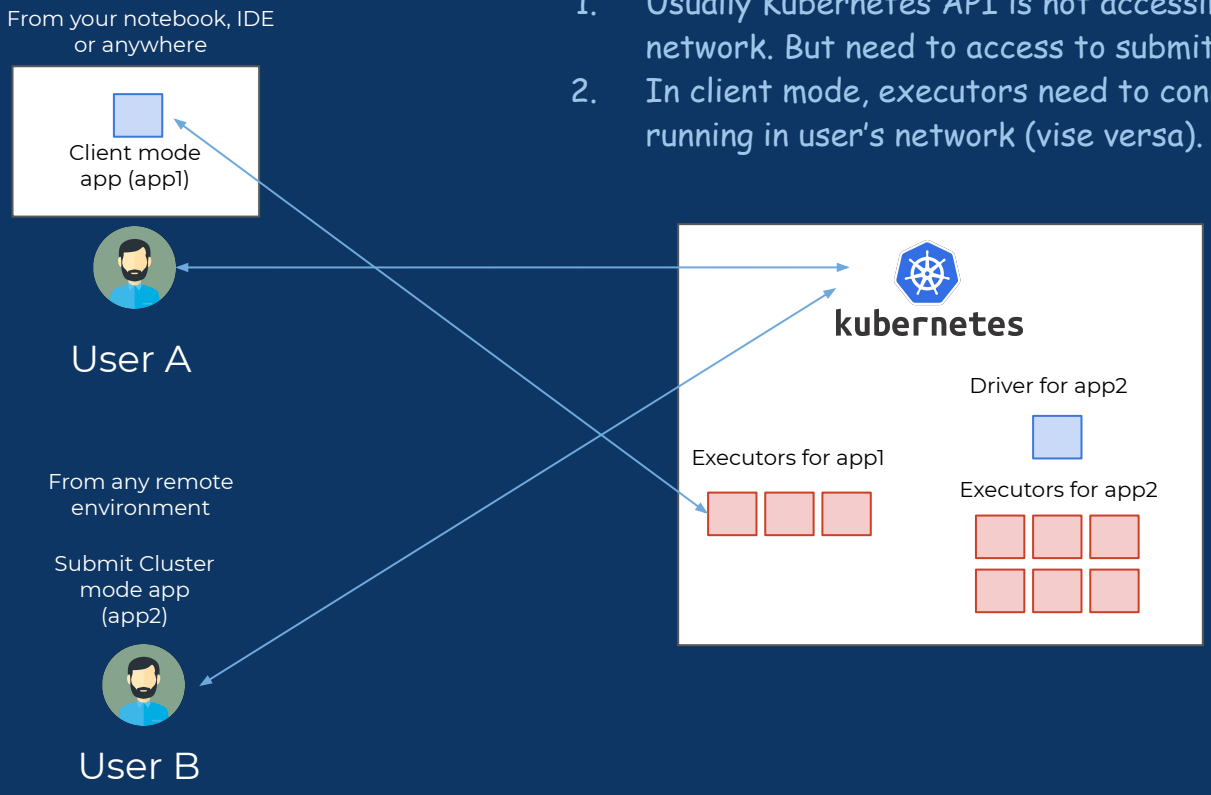
User B

Run my job!

1. Drivers / Executors are free to connect other apps's one
2. Executors can run arbitrary code from user. Executors are running inside container. Container does not provide strong isolation/security.
3. What if one user try to use too much resources?



Problem 2 - Connection



1. Usually Kubernetes API is not accessible from user's network. But need to access to submit Spark application
2. In client mode, executors need to connect to the driver running in user's network (vise versa).

STAROID

Solution1

1. Drivers / Executors are free to connect other apps's one

Create a network policy and only allow communication between the same group of driver and executors.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-only-same-namespace
  namespace: kubernetes-app1
spec:
  podSelector: {}
  ingress:
  - namespaceSelector:
      matchLabels:
        spark-serverless: kubernetes-app1
  policyTypes:
  - Ingress
```

Solution1

2. Executors can run arbitrary code from user. Executors are running inside container. Container does not provide strong isolation/security.

- Apply Pod security policy and force non-root container

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
 name: kubernetes-psp
spec:
 privileged: false  # Don't allow privileged pods!
 allowPrivilegeEscalation: false
 hostNetwork: false
 hostPorts:
 - min: 58856 # for dedicated vm
   max: 58856
 hostIPC: false
 hostPID: false
 readOnlyRootFilesystem: false
 runAsUser:
   # Require the container to run without root privileges.
   rule: 'MustRunAsNonRoot'
 runAsGroup:
   rule: 'MustRunAs'
   ranges:
   - min: 1
     max: 65535
```

Solution1

2. Executors can run arbitrary code from user. Executors are running inside container. Container does not provide strong isolation/security.

- Dedicated VM allocation per executor to take advantage of VM level isolation instead of container level isolation. (e.g. prevent https://meltdownattack.com/).

Implement Kubernetes mutating admission webhook that add hostPort to executor Pod when user marked Pod to be run on dedicated VM (via label or annotation)

mutating admission webhook

2. Admission webhook add hostPort when label is detected

1. Add a label to executor Pod using spark configuration

```
spark.kubernetes.executor.label.pod.staroid.com/isolation  dedicated
```

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: spark
    ports:
    - hostPort: 58856
```

3. Can not allocate the same hostPort on the same VM. therefore each Pod will be allocated to the different VM

STAROID

Solution1

3. What if one user try to use too much resources?
   Create a namespace for each application (set of driver and executors) and apply Resource Quota.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.cpu: "10"
    requests.memory: 100Gi
    limits.cpu: "20"
    limits.memory: 200Gi
```

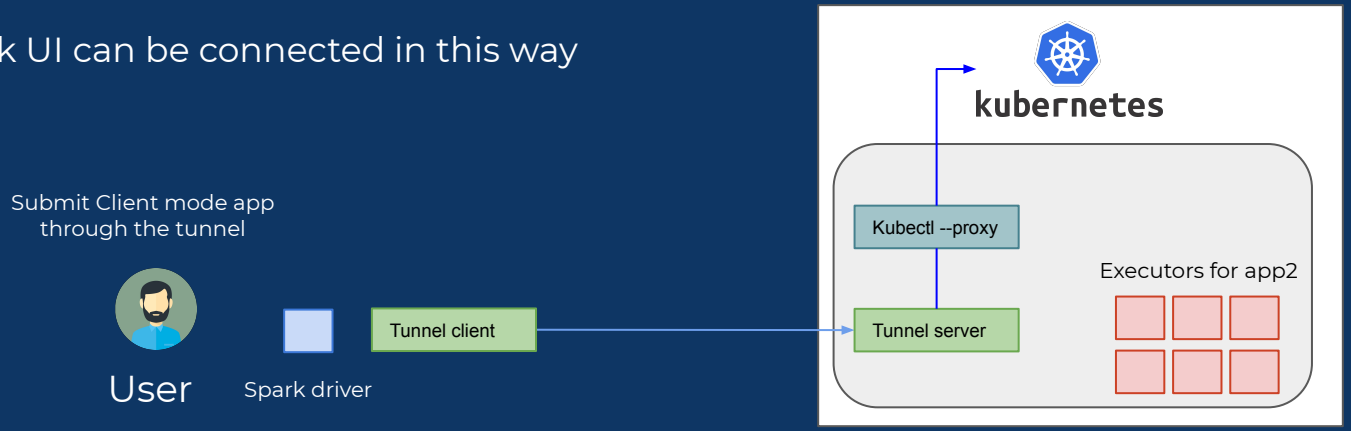
Solution2

1. Usually Kubernetes API is not accessible from user's network. But need to access to submit Spark application

Configure RBAC of each Namespace's service account (default) to have restricted permissions within the Namespace. And run tunnel server in the Kubernetes cluster. Deploy a Pod that runs "kubectl --proxy" in the target namespace.

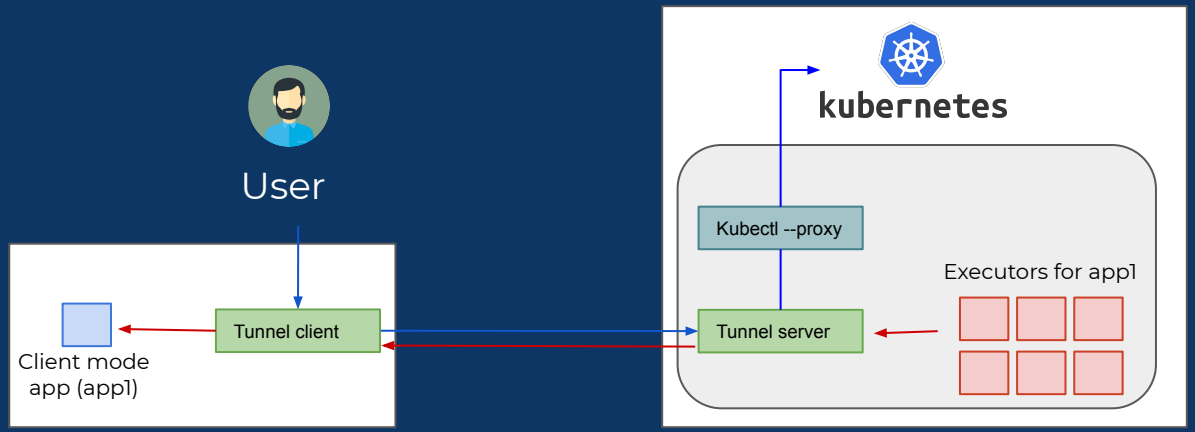
User can run Spark application just like user can do inside the Kubernetes cluster.

Spark UI can be connected in this way



Solution2

2. In client mode, executors need to connect to the driver running in user's network (vise versa).
In client mode, tunnel server provides a reverse tunnel as well so executors can connect to the driver running in user's environment.



STAROID

Spark Configuration

```
# dynamic allocation
spark.dynamicAllocation.enabled                         true
spark.dynamicAllocation.minExecutors                    1
spark.dynamicAllocation.maxExecutors                    10
spark.dynamicAllocation.initialExecutors                1
spark.dynamicAllocation.executorIdleTimeout             600s
spark.dynamicAllocation.schedulerBacklogTimeout         60s

# Kubernetes, there's no external shuffle service. so,
spark.dynamicAllocation.shuffleTracking.enabled         true

# larger batch size is helpful otherwise Kubernetes will provision node one by one when scaling out
spark.kubernetes.allocation.batch.size                  20

# spark 3.0 performance improvement
spark.sql.adaptive.enabled                              true
spark.sql.adaptive.coalescePartitions.enabled           true
```

STAROID

Client library

https://github.com/open-datastudio/ods

1. Download appropriate version of Spark binary locally
2. Configure Spark
   a. Connect through (reverse) tunnel
   b. Configure container image to choose compatible python version
3. Initiate (reverse) tunnel between kubernetes namespace and python environment
4. Create Spark session

STAROID

Pyspark in serverless

- Driver python environment and executor python environment can be different
  - E.g. Driver - python 3.6, Executor image - python 3.7
- Driver need to detect its python version and run executor in a same python version

```
PYSPARK_PYTHON=<executor python env path>
PYSPARK_DRIVER_PYTHON=<driver python env path>
```

https://github.com/open-datastudio/ods/blob/v0.0.7/ods/spark_cluster/spark_cluster.py#L238
https://github.com/open-datastudio/spark/blob/v3.1.0-snapshot-20200720-01/.github/workflows/publish-docker-image.yml#L51

Container image

- Includes multiple versions of Python environment (3.6, 3.7, 3.8)

Container image

```
50          cat <<EOT >> /tmp/Dockerfile
51          RUN cd /home/spark/ && curl https://pyenv.run | bash && \
52              /home/spark/.pyenv/bin/pyenv install 3.6.9 && \
53              /home/spark/.pyenv/bin/pyenv install 3.7.7 && \
54              /home/spark/.pyenv/bin/pyenv install 3.8.1 && \
55              /home/spark/.pyenv/bin/pyenv global 3.6.9 && \
56              rm -rf /tmp/python-build*
57          EOT
```

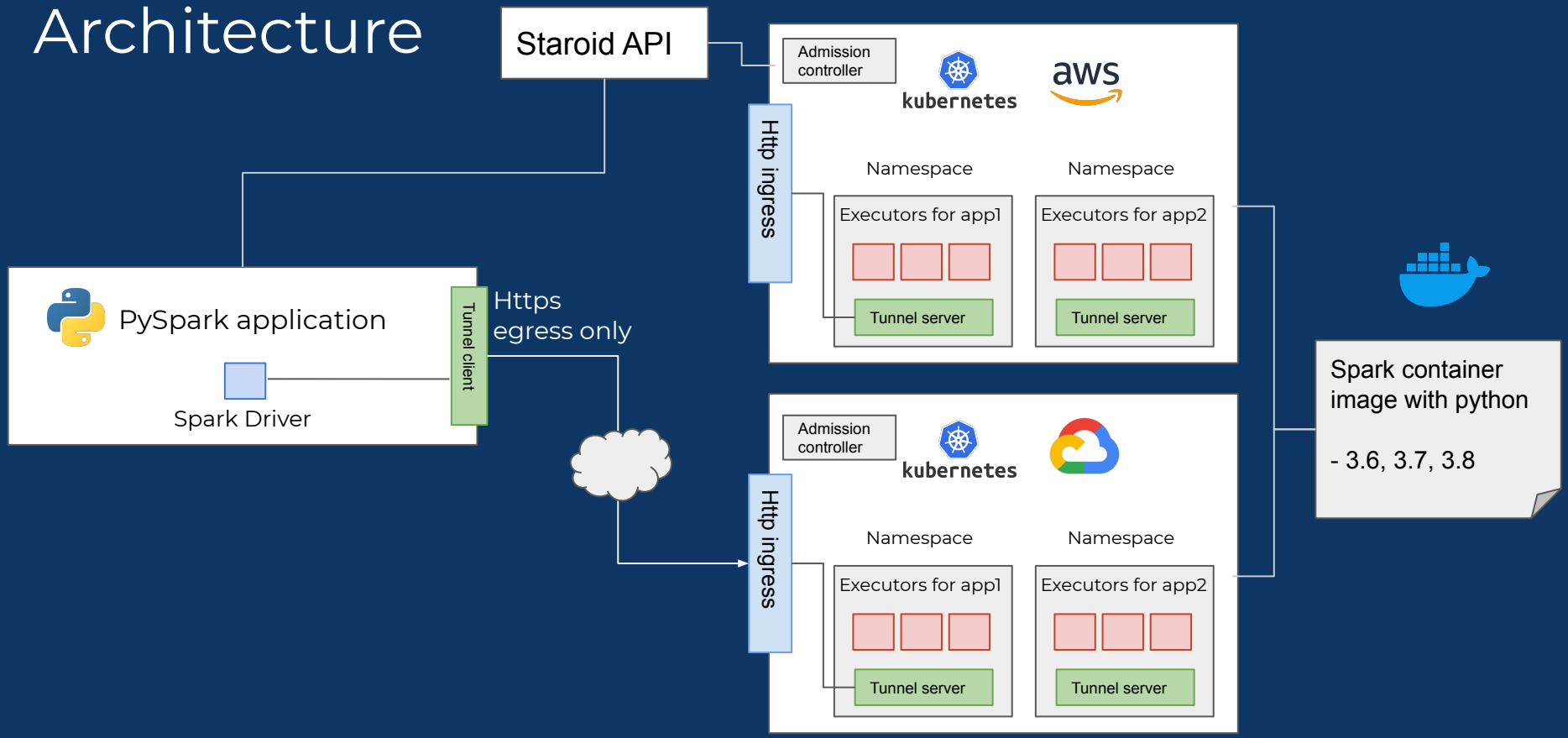
https://github.com/open-datastudio/spark/blob/master-staroid/.github/workflows/publish-docker-image.yml#L47

Client library

```
237         executor_python_path = None
238         if sys.version_info >= (3, 8) and sys.version_info < (3, 9):
239             executor_python_path = SPARK_IMAGE_PYTHON_PATH["3.8"]
240         elif sys.version_info >= (3, 7) and sys.version_info < (3, 8):
241             executor_python_path = SPARK_IMAGE_PYTHON_PATH["3.7"]
242         elif sys.version_info >= (3, 6) and sys.version_info < (3, 7):
243             executor_python_path = SPARK_IMAGE_PYTHON_PATH["3.6"]
244         else:
245             raise Exception("Current python version is not supported. Su
246
247         os.environ["PYSPARK_PYTHON"] = executor_python_path
248         os.environ["PYSPARK_DRIVER_PYTHON"] = sys.executable
```

https://github.com/open-datastudio/ods/blob/master/ods/spark_cluster/spark_cluster.py

Architecture



STAROID

Conclusions

- Easy to use Spark cluster from various environments
  - (e.g. notebook environment running on data scientist's laptop)
- Better resource utilization
  - Executors are created when needed. Terminated when jobs are done.
- Better security / isolation
  - Each application get their own Spark cluster (set of executors)
- Fast spin-up
  - Initial Spinup time takes few seconds to 1-2 minutes (in case new node need to be provisioned in Kubernetes cluster)

STAROID

Staroid

https://staroid.com

- Cloud platform

  For open source developers and enterprise users
- Runtime based on Kubernetes
- StarRank

  Fund developers when user deploy the project
- Enterprise Support

Open data studio

https://open-datastudio.io

- Open source!
- Releases service on the cloud

  Instead of release as binary/source package
- Focuses on data processing / Machine learning softwares

  e.g. Spark, Ray, and so on
- Let’s build it together

STAROID

Implementation & reference

Spark serverless project
https://github.com/open-datastudio/spark-serverless

Python Client
https://github.com/open-datastudio/ods

Spark Docker image
https://github.com/open-datastudio/spark/blob/master-staroid/.github/workflows/publish-docker-image.yml#L36

Documentation
http://open-datastudio.io/computing/spark/index.html

Tunnel, Reverse tunnel
https://github.com/jpillora/chisel

Thanks

Moon soo Lee / moon@staroid.com