Setting up predictive analytics services with Palladium

Andreas Lattner
Data Science Team, Business Intelligence, Otto Group
PyData Berlin, May 20, 2016

otto group

Agenda

1. Introduction
2. Architecture
3. Example: Setting up a classification service
4. Deployment with Docker and Mesos / Marathon
5. Summary

otto group

Motivation & History

Palladium emerged from a project for parcel delivery time prediction for Hermes

- Frequent development of predictive analytics prototypes (R, Python, …)
- Partly reimplementation to set up operational applications
- A great stack of data analysis and machine learning packages exist for Python (numpy, scipy, pandas, scikit-learn, …)

Requirements Predictive Analytics

- Reduce transition time from prototypes to operational systems
- High scalability
- Provide reliable predictive services

+

- Avoid expenses for licenses
- Faster start of projects, avoid re-implementation of same functionality

Parcel Delivery Time Prediction Framework

Generic solution for group‘s services

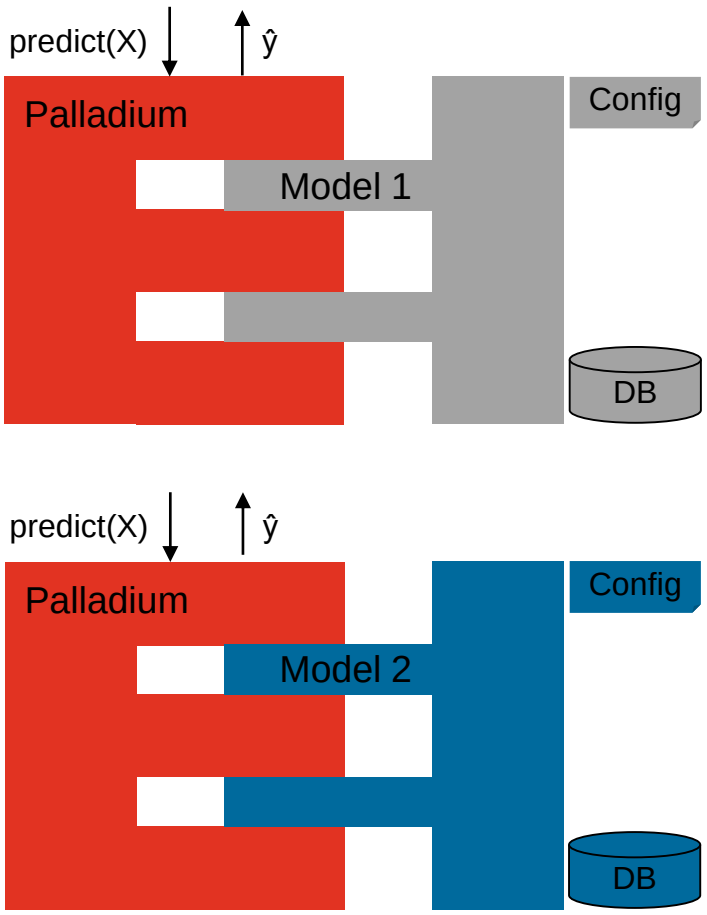
PALLADIUM

otto group

What is Palladium?

Framework for fitting, evaluating, saving, deploying and using (predictive) models

- **Unified model management (Python, R, Julia)**
  Palladium allows fitting, storing, loading, distributing, versioning of models; metadata management; using scikit-learn's interfaces
- **Generation of operative predictive services**
  Provisioning of models as web services
- **Flexibility**
  It is possible to quickly set up new services via configuration and interfaces
- **Automated update**
  Update of a service's data / models in configurable intervals
- **Scalability**
  Easy distribution and scalability of predictive services via Docker containers and integration to load balancer

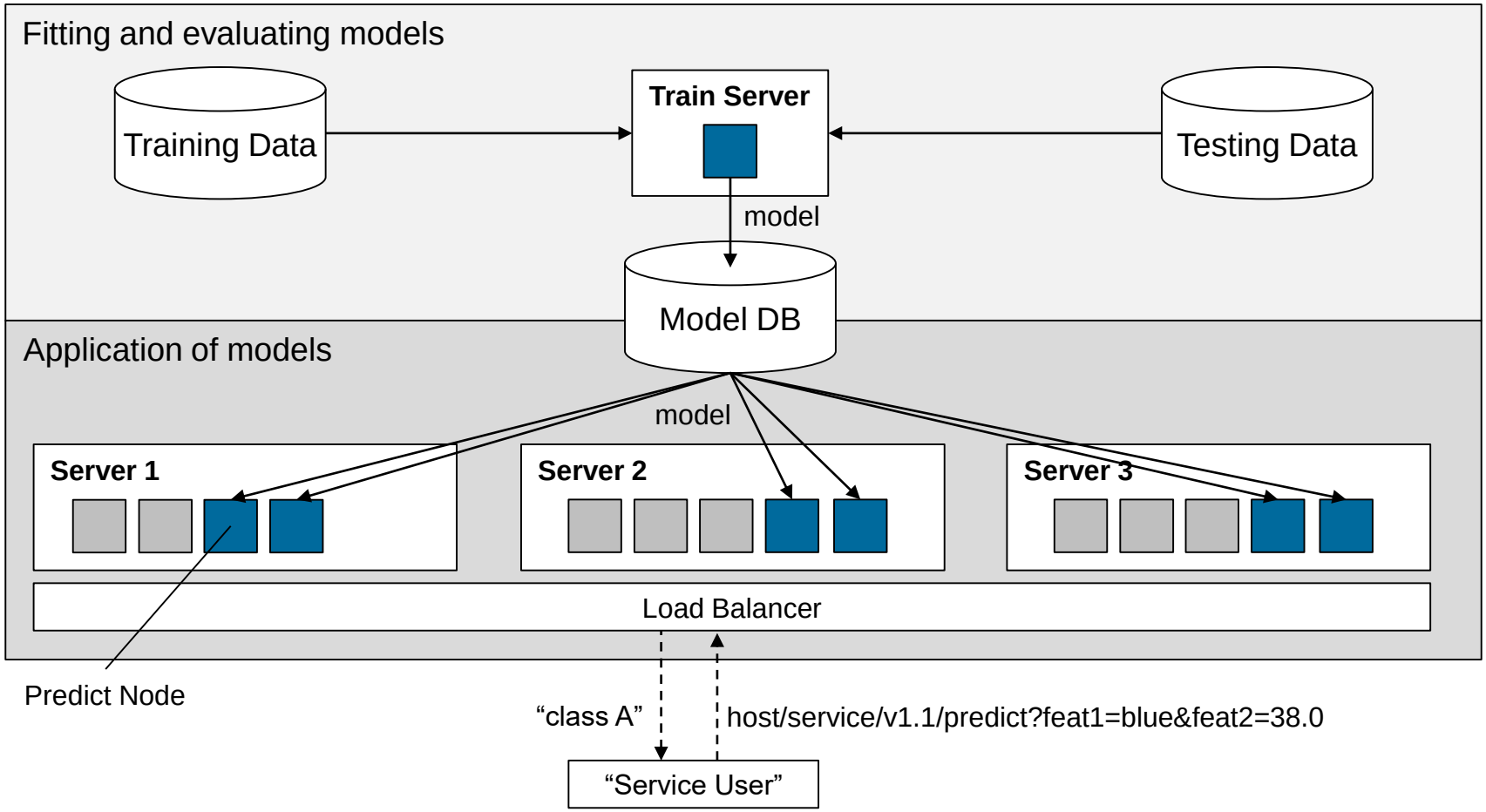


Agenda

1. Introduction
2. Architecture
3. Example: Setting up a classification service
4. Deployment with Docker and Mesos / Marathon
5. Summary

otto group

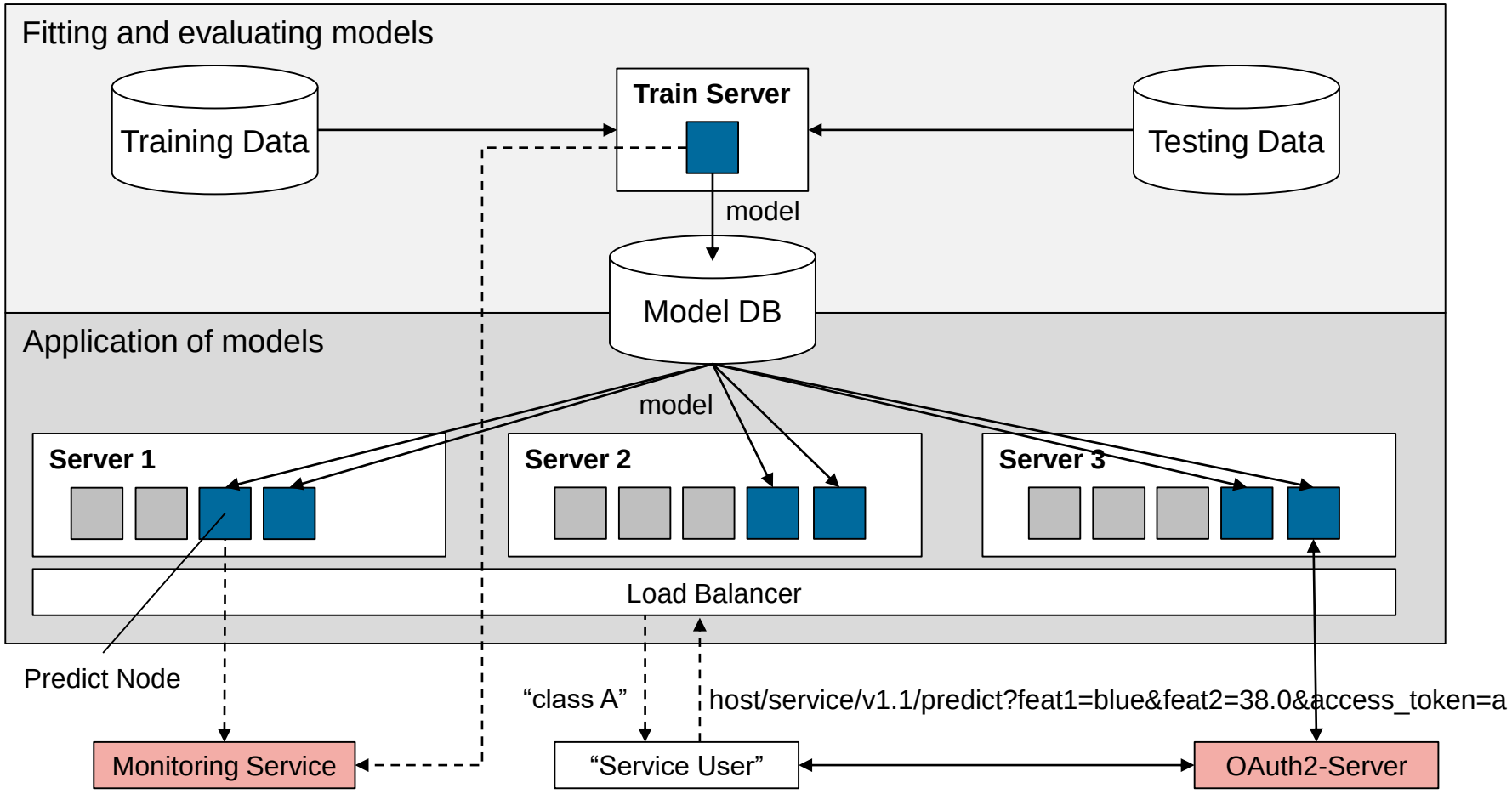
Architecture



otto group

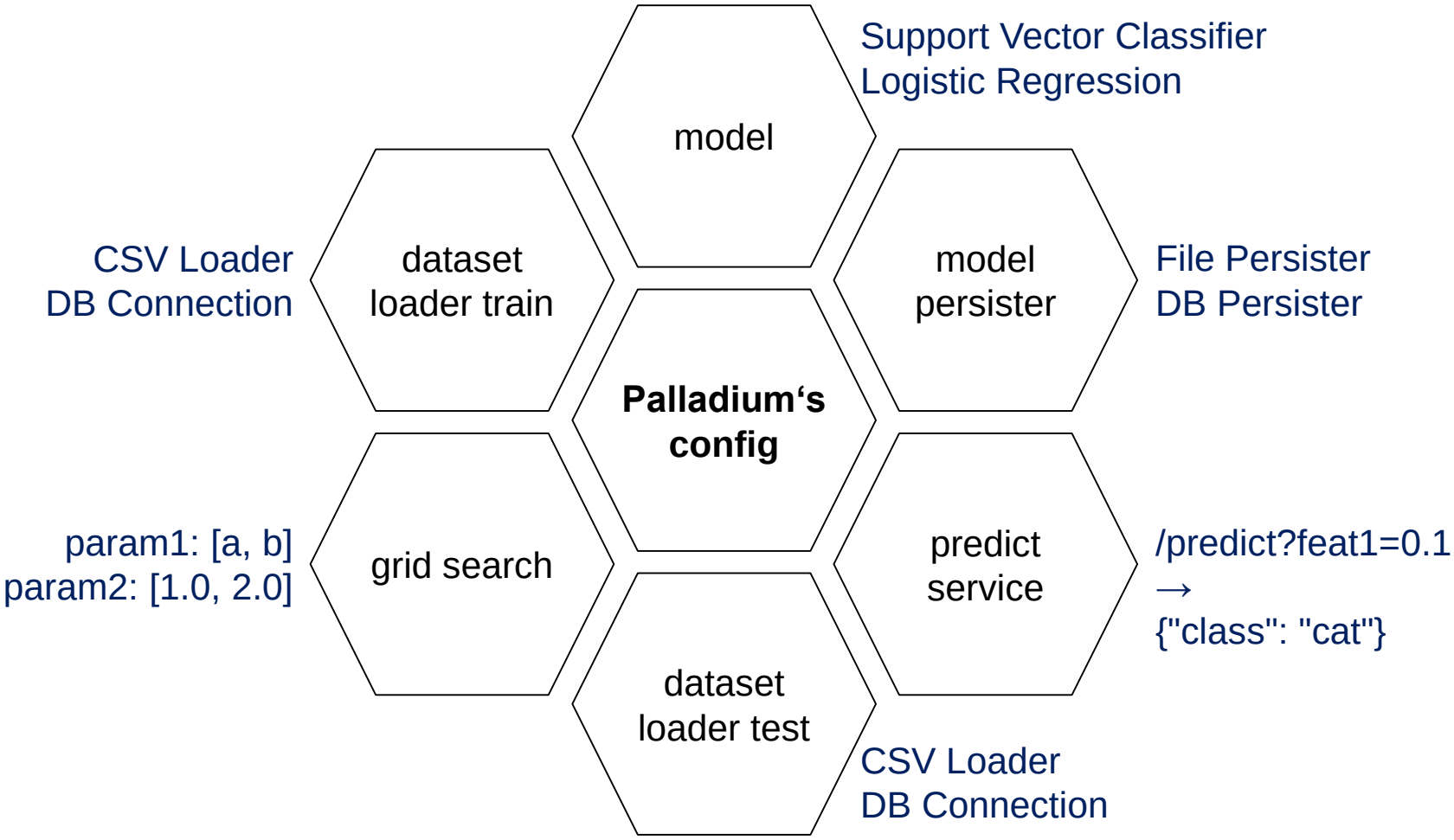
Architecture

Flexible integration of Authentication, Logging and Monitoring



otto group

Flexible Structure via Interfaces



otto group

Agenda

1. Introduction
2. Architecture
3. **Example: Setting up a classification service**
4. Deployment with Docker and Mesos / Marathon
5. Summary

otto group

Example: Iris Classification

```
sepal length:  5.2
sepal width:   3.5
petal length:  1.5
petal width:   0.2
```

```
Iris-setosa
Iris-versicolor
Iris-virginica
```

?

Training & test data

```
5.2,3.5,1.5,0.2,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.6,3.0,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.1,3.8,1.5,0.3,Iris-setosa
...
```

```
http://localhost:5000/predict?
sepal length=5.2&sepal width=3.5&
petal length=1.5&petal width=0.2
```

```
{"result": "Iris-virginica",
 "metadata":{
    "service_name": "iris",
    "service_version": "0.1",
    "error_code": 0,
    "status": "OK"}}
```

Palladium Predict Server

otto group

Configuration and Corresponding Classes

```
'dataset_loader_train': {...},
'dataset_loader_test': {...},
'model': {...},
'grid_search': {...},
'model_persister': {...},
'predict_service': {...},
```

DatasetLoader

DatasetLoader

Model (→sklearn.base.BaseEstimator)

sklearn.grid_search.GridSearchCV

ModelPersister

PredictService

otto group

Configuration

```
'dataset_loader_train': {...},

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

'predict_service': {...},
```

```
'__factory__':
    'palladium.dataset.Table',
    'path': 'iris.data',
    'names': [
        'sepal length',
        'sepal width',
        'petal length',
        'petal width',
        'species',
    ],
    'target_column': 'species',
    'sep': ',',
    'nrows': 100,
```


otto group

Configuration

```
'dataset_loader_train': {...},

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

'predict_service': {...},
```

```
'__factory__':
    'palladium.dataset.Table',
    'path': 'iris.data',
    'names': [
        'sepal length',
        'sepal width',
        'petal length',
        'petal width',
        'species',
    ],
    'target_column': 'species',
    'sep': ',',
    'skiprows': 100,
```


otto group

Configuration

```
'dataset_loader_train': {...},

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

'predict_service': {...},
```

```
'__factory__':
    'sklearn.tree.
     DecisionTreeClassifier',
    'min_samples_leaf': 1,
```


otto group

Configuration

```
'dataset_loader_train': {...},

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

'predict_service': {...},
```

```
'param_grid': {
    'min_samples_leaf':
        [1, 2, 3],
},
'verbose': 4,
'n_jobs': -1,
```


otto group

Configuration

```
'dataset_loader_train': {...},

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

'predict_service': {...},
```

```
'__factory__':
    'palladium.persistence.File',
'path':
    'iris-model-{version}',
```


otto group

Configuration

```
'dataset_loader_train': {...},

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

'predict_service': {...},
```

```
'__factory__':
    'palladium.server.
     PredictService',
'mapping': [
    ('sepal length', 'float'),
    ('sepal width', 'float'),
    ('petal length', 'float'),
    ('petal width', 'float'),
    ],
```


otto group

Fitting and Testing Models

- Script for fitting models: pld-fit
  - Loads training data
  - Fits model (using specified estimator)
  - Stores model + metadata
- Option to evaluate model on validation set (--evaluate)

```
INFO:palladium:Loading data...
INFO:palladium:Loading data done in 0.010 sec.
INFO:palladium:Fitting model...
INFO:palladium:Fitting model done in 0.001 sec.
INFO:palladium:Writing model...
INFO:palladium:Writing model done in 0.039 sec.
INFO:palladium:Wrote model with version 8.
```

Fitting and Testing Models

- Script for testing different parameters: pld-grid-search
  - Loads training data
  - Splits training data in folds (cross validation)
  - Creates runs for all parameter-fold combinations
  - Reports results for different settings

```
INFO:palladium:Loading data...
INFO:palladium:Loading data done in 0.004 sec.
INFO:palladium:Running grid search...
Fitting 3 folds for each of 3 candidates, totalling 9 fits
...
[Parallel(n_jobs=-1)]: Done 9 out of 9 | elapsed: 0.1s finished
INFO:palladium:Running grid search done in 0.041 sec.
INFO:palladium:
[mean: 0.93000, std: 0.03827, params: {'min_samples_leaf': 2},
 mean: 0.92000, std: 0.02902, params: {'min_samples_leaf': 1},
 mean: 0.92000, std: 0.02902, params: {'min_samples_leaf': 3}]
```


otto group

Fitting and Testing Models

- Script for testing models: pld-test
  - Loads test data
  - Applies model to test data
  - Reports results (e.g., accuracy)

```
INFO:palladium:Loading data...
INFO:palladium:Loading data done in 0.003 sec.
INFO:palladium:Reading model...
INFO:palladium:Reading model done in 0.000 sec.
INFO:palladium:Applying model...
INFO:palladium:Applying model done in 0.001 sec.
INFO:palladium:Score: 0.92.
```

Deploying and Applying Models

- Built-in script for providing models: pld-devserver
  - Using Flask's web server
- Recommended to use WSGI container / web server, e.g., gunicorn / nginx

- Predict server
  - Loads model (model persister)
  - Schedule for model updates
  - Provides web service entry points ("predict", "alive")

/alive


```
{
  ▾ model: {
        updated: "2016-05-19T13:28:27.729214",
      ▾ metadata: {
            train_timestamp: "2016-05-19T13:26:48.929500",
            score_train: 0.94,
            score_test: 0.96,
            version: 2
        }
    },
  ▾ service_metadata: {
        service_name: "iris",
        service_version: "0.1"
    },
    memory_usage_vms: 524.7578125,
    palladium_version: "1.0.1",
    memory_usage: 92.875
}
```


/predict


```
{
  ▾ metadata: {
        error_code: 0,
        service_name: "iris",
        service_version: "0.1",
        status: "OK"
    },
    result: "Iris-virginica"
}
```


otto group

Testing Service Overhead (1 CPU)

Including prediction of Iris model; using Flask‘s develop server

```
ab -n 1000
"http://localhost:4999/predict?sepal%20length=5.2&sepal%20width=
3.5&petal%20length=1.5&petal%20width=0.2"


Time taken for tests:   1.217 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      273000 bytes
HTML transferred:       112000 bytes
Requests per second:    821.82 [#/sec] (mean)
Time per request:       1.217 [ms] (mean)
Time per request:       1.217 [ms] (mean, across all concurrent
requests)
Transfer rate:          219.10 [Kbytes/sec] received
```

(Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz)

otto group

Extensions

- Dynamic instantiation of objects (not only for provided interfaces)
  - “__factory__” entries are instantiated on config initialization (using resolve_dotted_name) and can be accessed via get_config()[‘name’]
  - Parameters are passed to constructor

```
'model': {
        '__factory__': 'sklearn.tree.DecisionTreeClassifier',
        'min_samples_leaf': 1,
        },
```

- Extension using decorators
  - A list of decorators can be set to wrap different calls (predict, fit, update model)
  - Can be used, e.g., for authentication or monitoring of “predict” calls

```
'predict_decorators': [
    'my_oauth2.authorization',
    'my_monitoring.log',
    ],
```


otto group

Extensions (2)

- Own implementation of PredictService can be set in config, e.g., to adapt response format or to define own way how sample is created from request

- Here we add a prediction_id to the response:

```
class MyPredictService(PredictService):
    def response_from_prediction(self, y_pred, single=True):
        result = y_pred.tolist()
        metadata = get_metadata()
        metadata.update({'prediction_id': str(uuid.uuid1())})

        if single:
            result = result[0]
        response = {
            'metadata': metadata,
            'result': result,
            }
        return make_ujson_response(response, status_code=200)
```


otto group

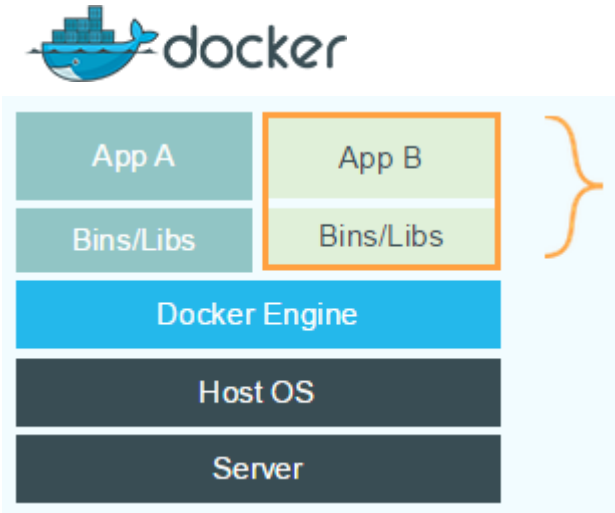
Agenda

1. Introduction
2. Architecture
3. Example: Setting up a classification service
4. **Deployment with Docker and Mesos / Marathon**
5. Summary

otto group

Docker, Mesos & Marathon

- Docker is a platform for the creation, distribution, and execution of applications
- Lightweight environment
- Easy combination of components
- Self-contained container including dependencies
- Docker registry for deployment



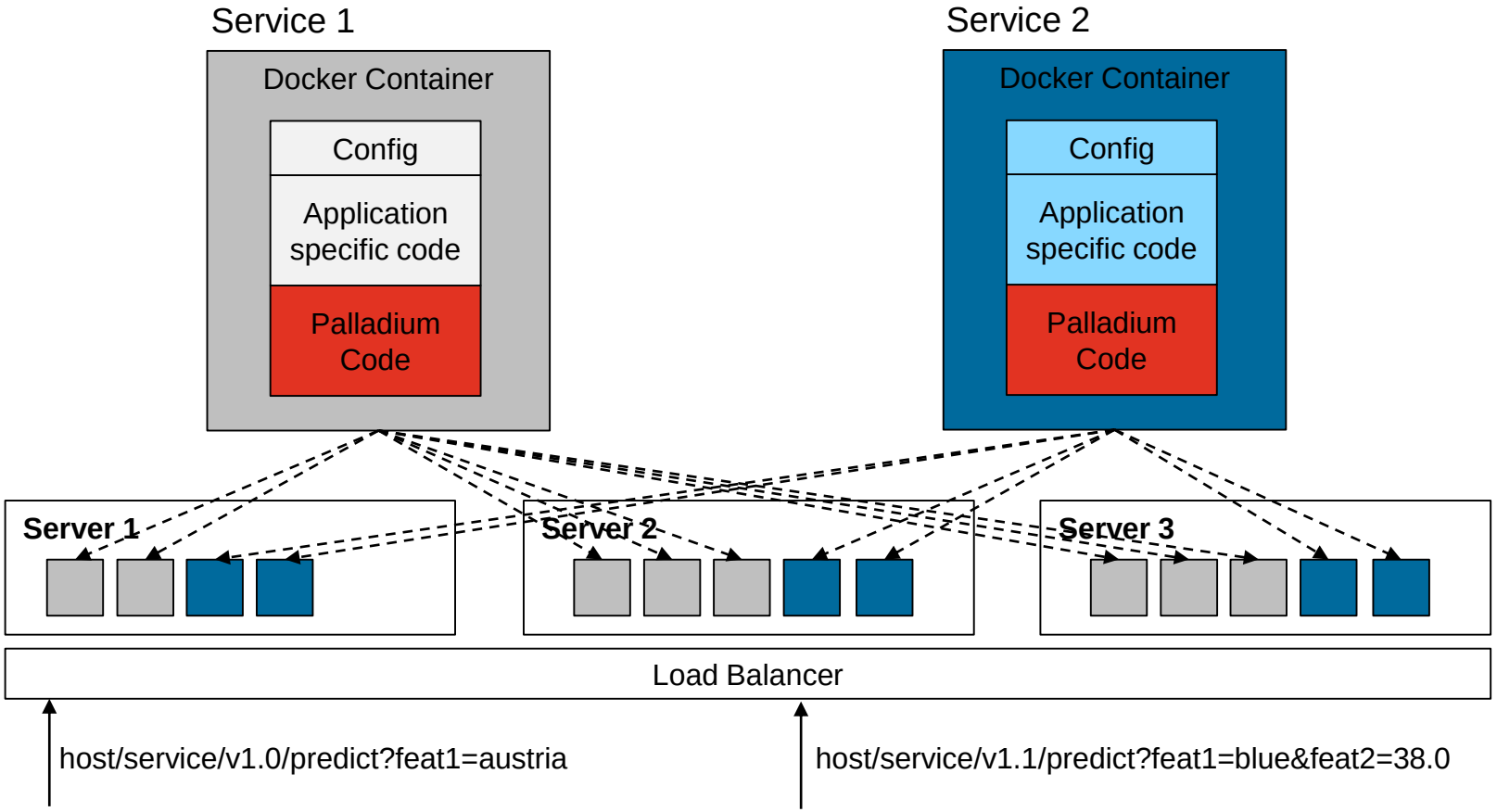
Source: https://www.docker.com/whatisdocker/

- Cluster framework Mesos provides resources, encapsulating details about used hardware
- High scalability and robustness
- Marathon (Mesosphere): Framework to launch and monitor services; used in combination with Mesos

Sources:
http://mesos.apache.org/
https://mesosphere.github.io/marathon/

otto group

Containers for and Deployment of Palladium Instances



otto group

Automated generation of Docker images

pld-dockerize

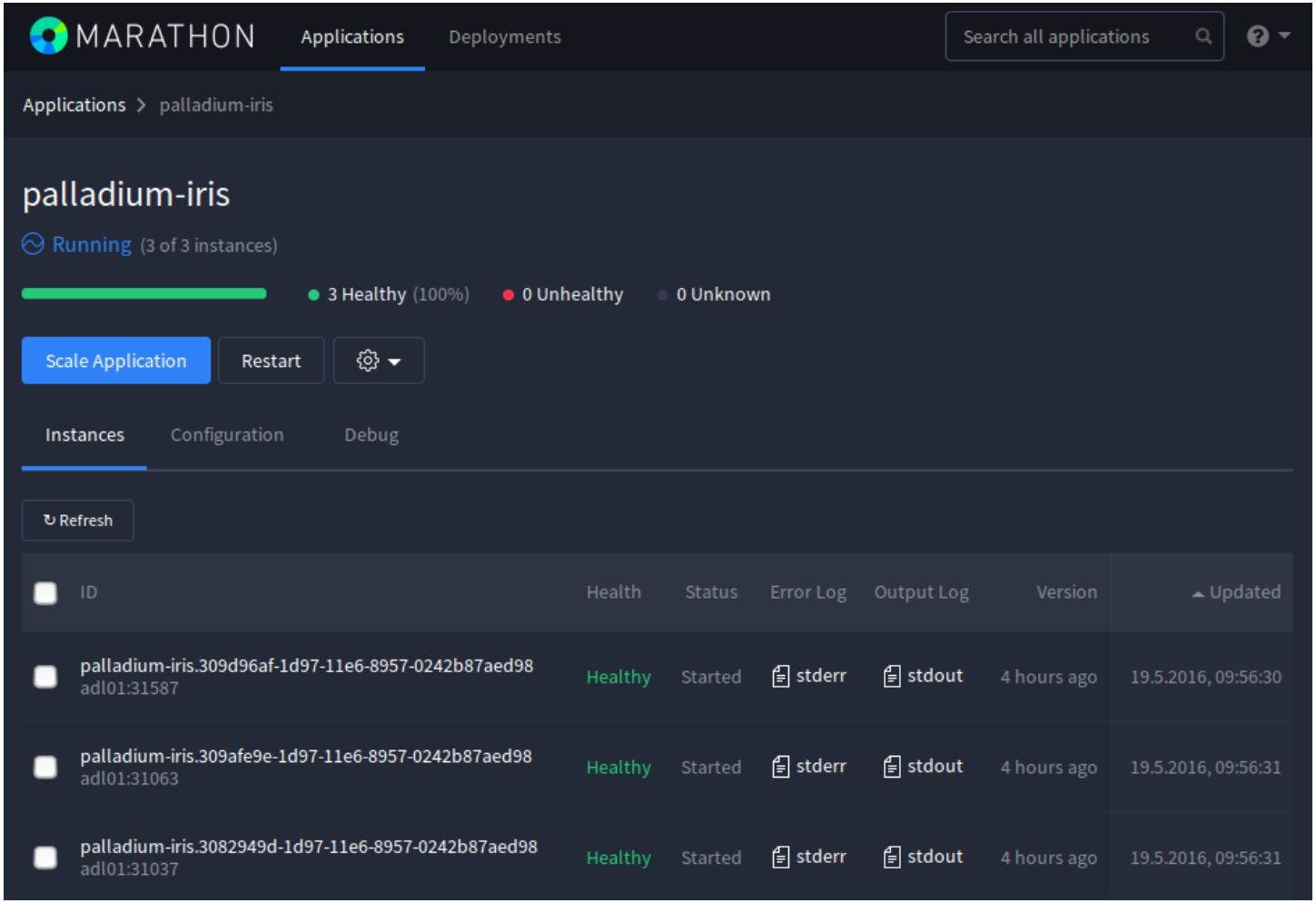
- Script pld-dockerize creates Docker image for predictive analytics service
- Example:
  pld-dockerize pld_codetalks ottogroup/palladium-base:1.0.1 alattner/iris-demo-tmp:0.1

- There exists an option to create only the Dockerfile without building the image (-d)

otto group

Deploying via Mesos / Marathon

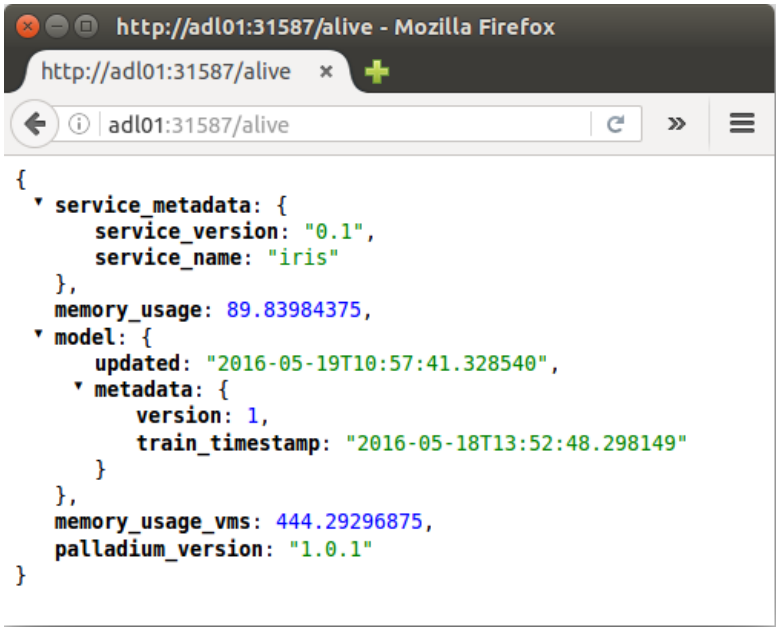
Easy deployment: Referring to Docker image and specifying number of instances



otto group

Deploying via Mesos / Marathon

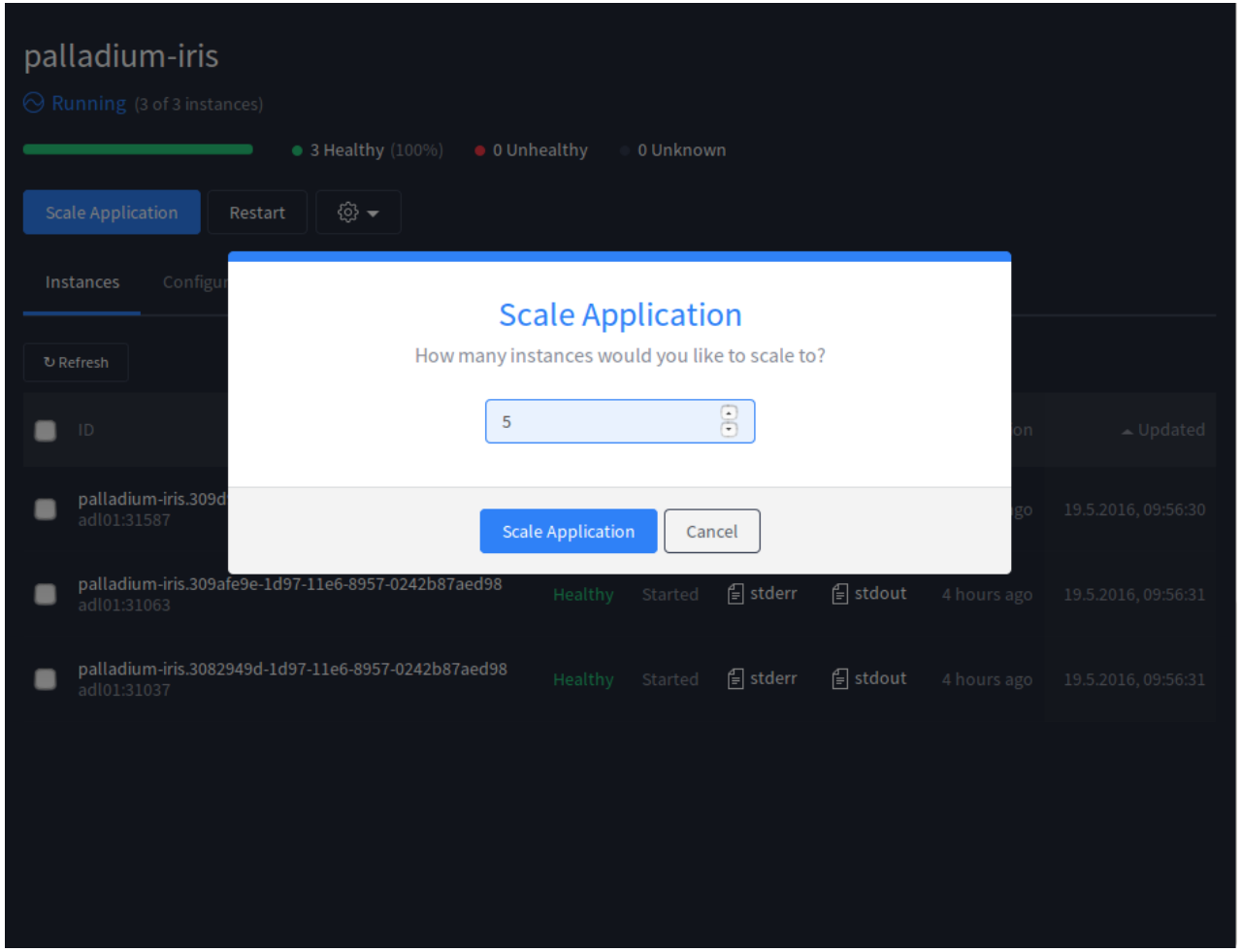
Palladium instances provide service after deployment



otto group

Deploying via Mesos / Marathon

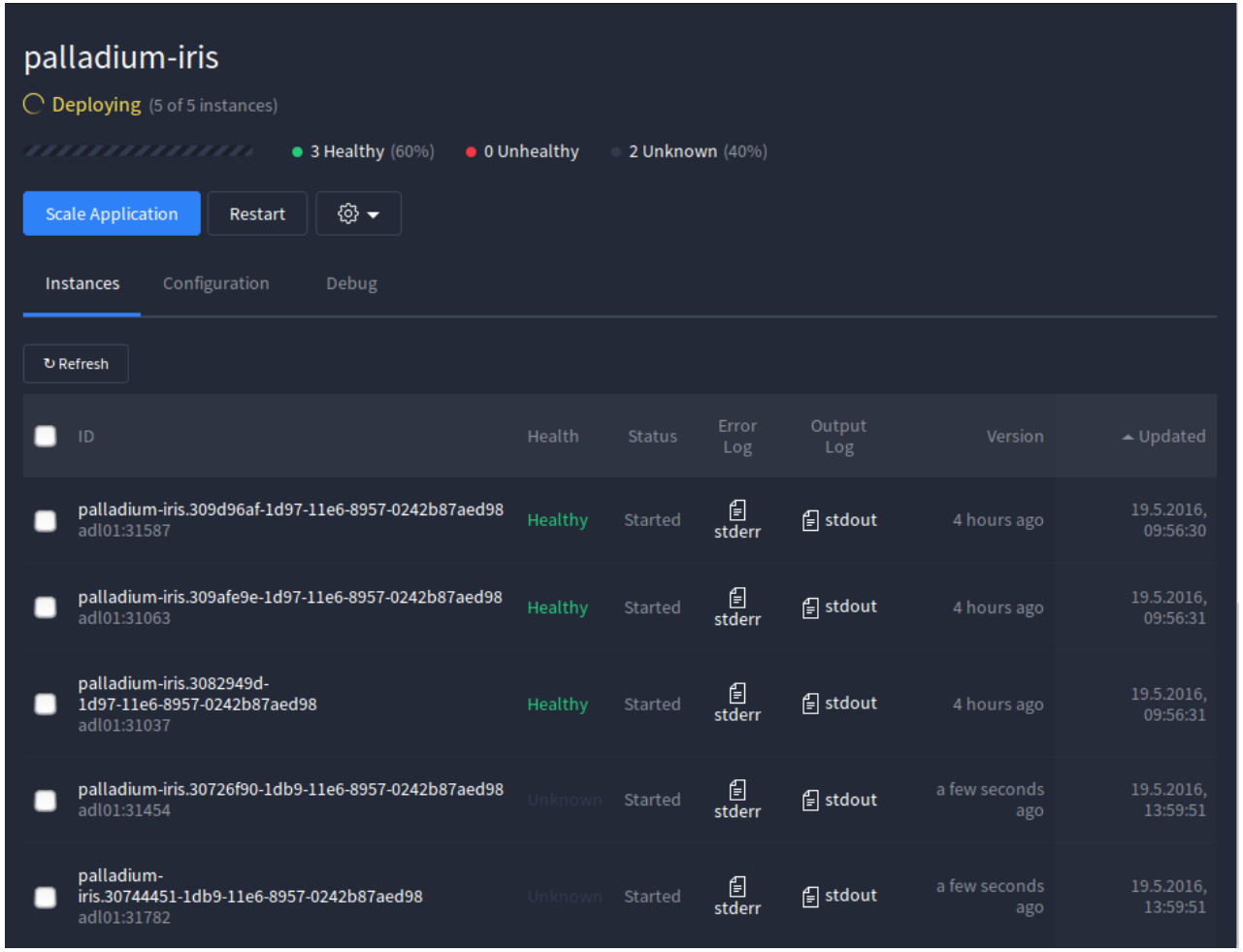
Easy scaling via GUI if more or less Palladium instances are desired



otto group

Deploying via Mesos / Marathon

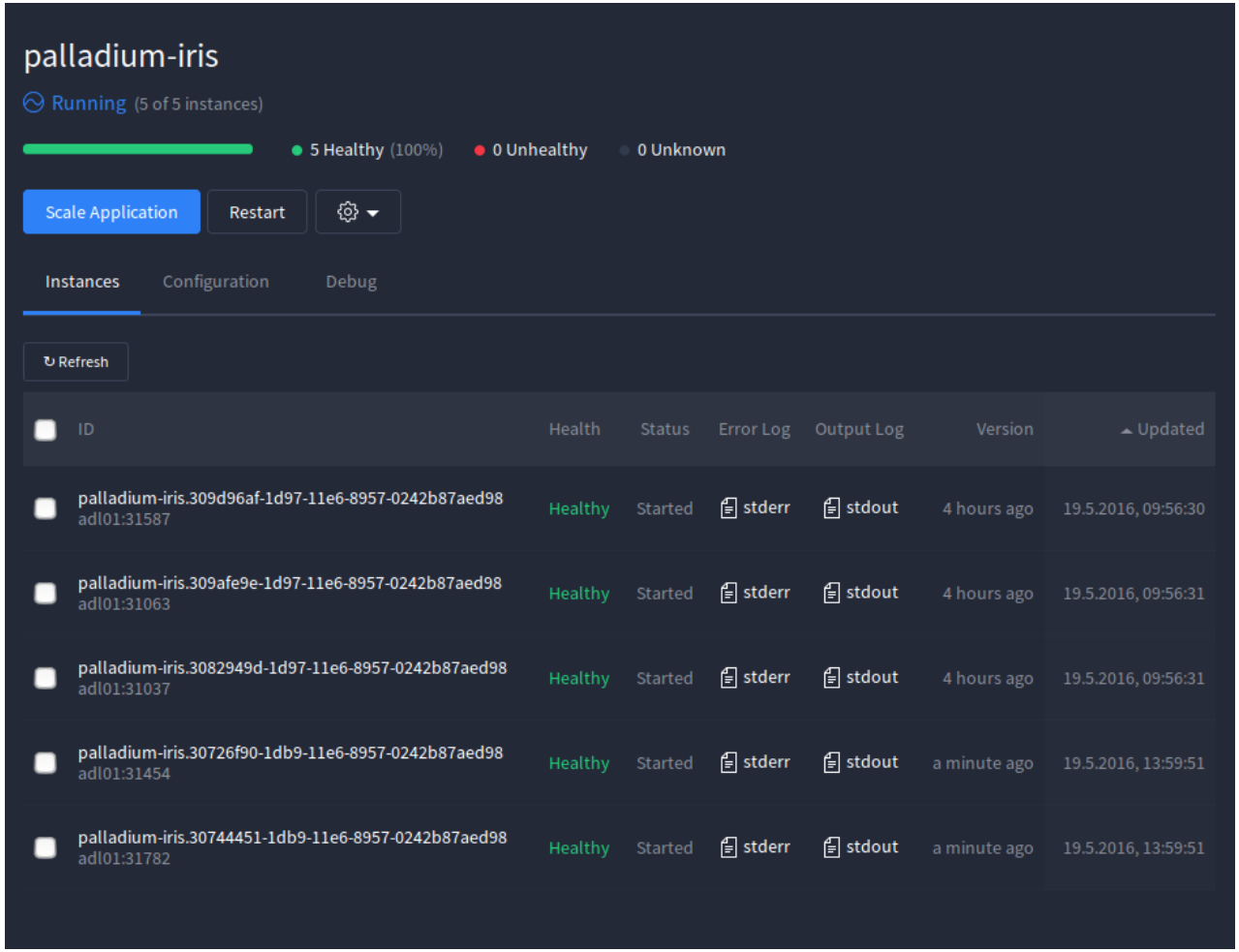
Easy scaling via GUI if more or less Palladium instances are desired



otto group

Deploying via Mesos / Marathon

Easy scaling via GUI if more or less Palladium instances are desired



otto group

Summary

- Palladium 1.0.1 is available at GitHub, PyPI, Anaconda (Linux)
- Easy way to expose ML models as web services using scikit-learn's interface
- Mechanism for automated update of models
- Script to automatically create Docker images for Palladium services
- Easy integration of other relevant services via decorator lists
  - Authentication
  - Logging, monitoring
- Support for models in other languages than Python: R (via rpy2), Julia (via pyjulia)
- Test-driven development, 100% test coverage
- Various Otto Group services have been realized with Palladium

- We'd be happy to receive feedback, suggestions for improvements, or pull requests!

otto group

Acknowledgment

- Daniel Nouri (design & development)
- Tim Dopke (Palladium + Docker, Mesos / Marathon)

- Data Science Team of the Otto Group BI

- Developers of used packages, e.g.,
  - scikit-learn
  - numpy
  - scipy
  - pandas
  - flask
  - sqlalchemy
  - pytest
  - …

Thank you very much for your attention!

otto group