Aufsetzen skalierbarer Prognose- und Analysedienste mit Palladium

Dr. Andreas Lattner
Group Business Intelligence, Otto Group

otto group

Agenda

1. Einleitung
2. Framework und Architektur
3. Beispiel: Aufsetzen eines Klassifikationsdienstes mit Palladium
4. Deployment mit Docker und Mesos / Marathon
5. Zusammenfassung

otto group

Historie und Motivation

Palladium basiert auf einem von der Otto Group BI bei Hermes entwickelten Kern

Anforderungen Predictive Analytics in der Gruppe

- Machine Learning Modelle schnell vom Prototyp in Produktion zu bringen. Beschleunigung Übergang Datenanalyse zu Produktion
- Hohe Skalierbarkeit
- Zuverlässiger Betrieb und Ausfallsicherheit bzgl. Hardwaredefekten

+

- Vermeidung von Lizenzkosten bei Gruppenunternehmen
- Schneller Projektstart. Das Rad nicht für jedes Predictive Analytics-Projekt neu erfinden

Prediction Framework für Zeitfensterprognose

Erweiterung für Group-weite Anwendung

PALLADIUM

otto group

Agenda

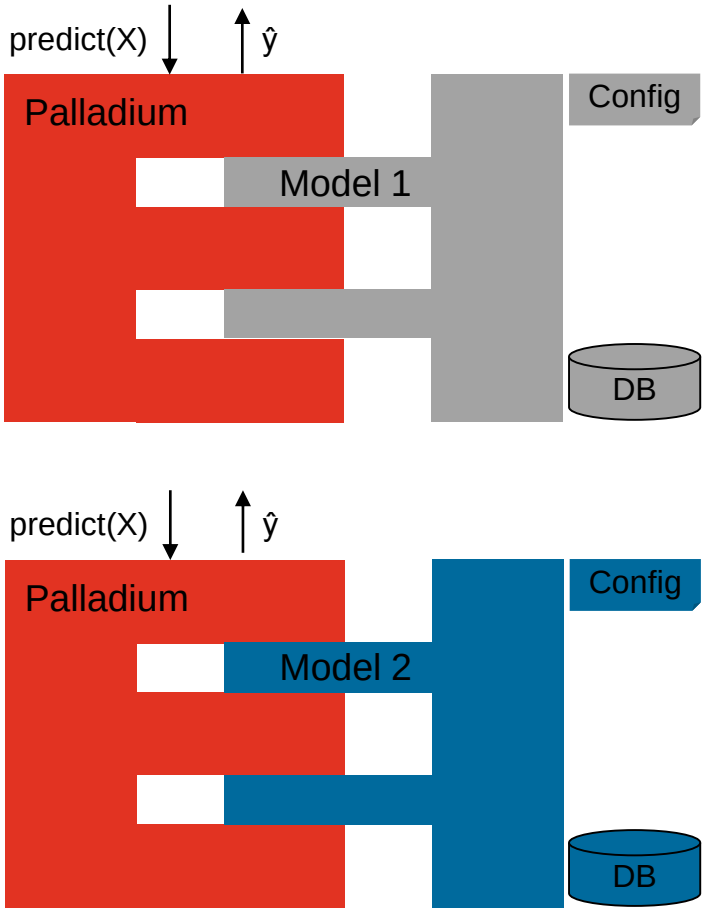
1. Einleitung
2. Framework und Architektur
3. Beispiel: Aufsetzen eines Klassifikationsdienstes mit Palladium
4. Deployment mit Docker und Mesos / Marathon
5. Zusammenfassung

otto group

Palladium: Framework für die Erzeugung, Evaluation, Verteilung und Nutzung von Prädiktionsmodellen

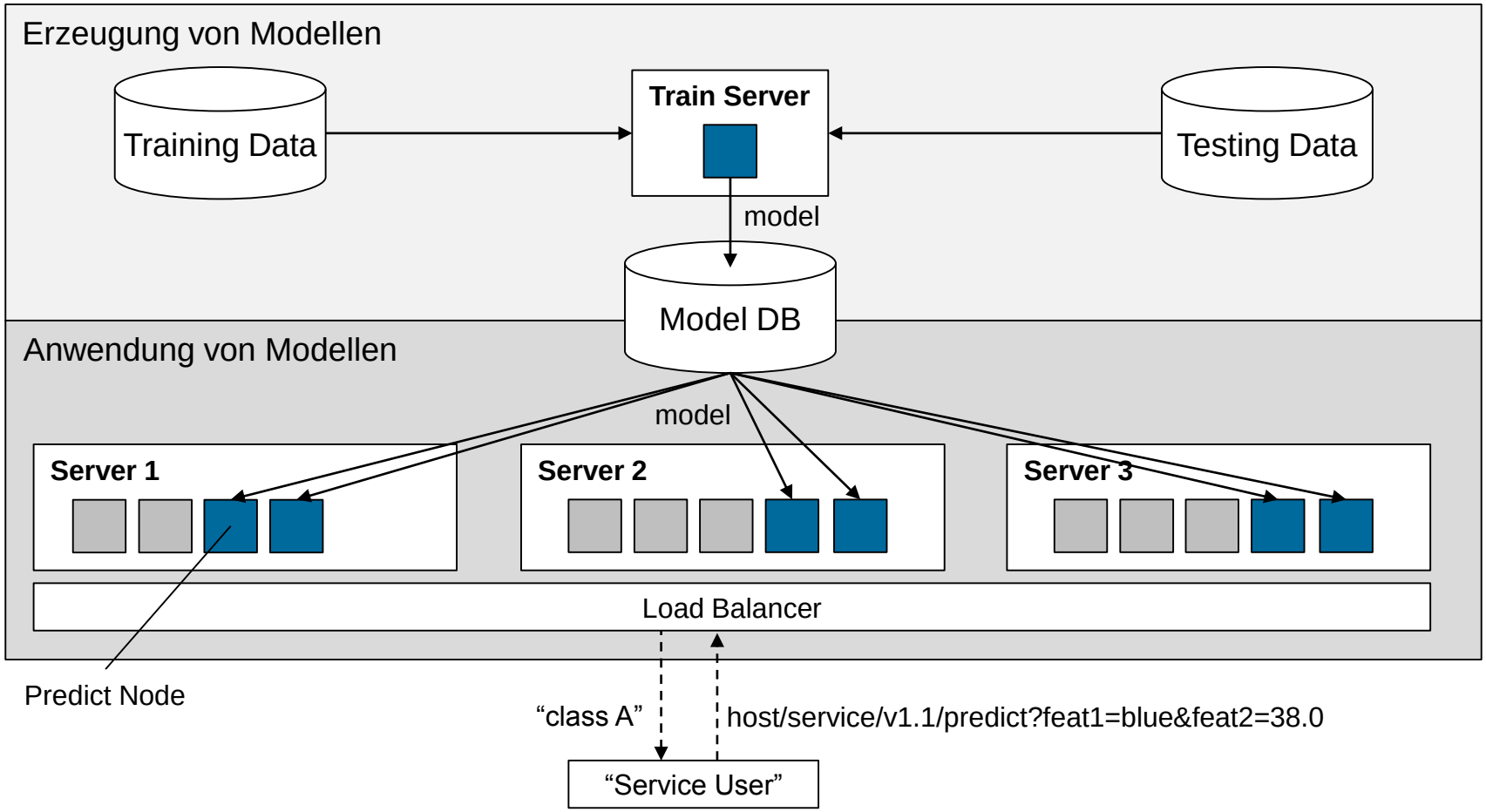
Anwendungsspezifische Aspekte lassen sich flexibel über Schnittstellen integrieren

- **Model Management (Python, R, Julia)**
  Palladium ermöglicht Erstellen, Speichern, Laden, Verteilen & Versionieren von Modellen, Erfassung von Metadaten
- **Erstellung operativer Prognosedienste**
  Bereitstellung der Dienste als Web Service
- **Flexibilität**
  Über Konfiguration und definierte Schnittstellen lassen sich schnell neue Dienste aufsetzen
- **Automatisierte Aktualisierung**
  Update der Daten / Modelle für Dienst in konfigurierbaren Zyklen
- **Skalierbarkeit**
  Einfaches Verteilen und Skalieren der Prognosedienste über Docker-Container und Load Balancer
- **Autorisierung und Logging**
  OAuth2-Integration und Logging für Prüfung der Zugriffsrechte und Nutzungsstatistik



otto group

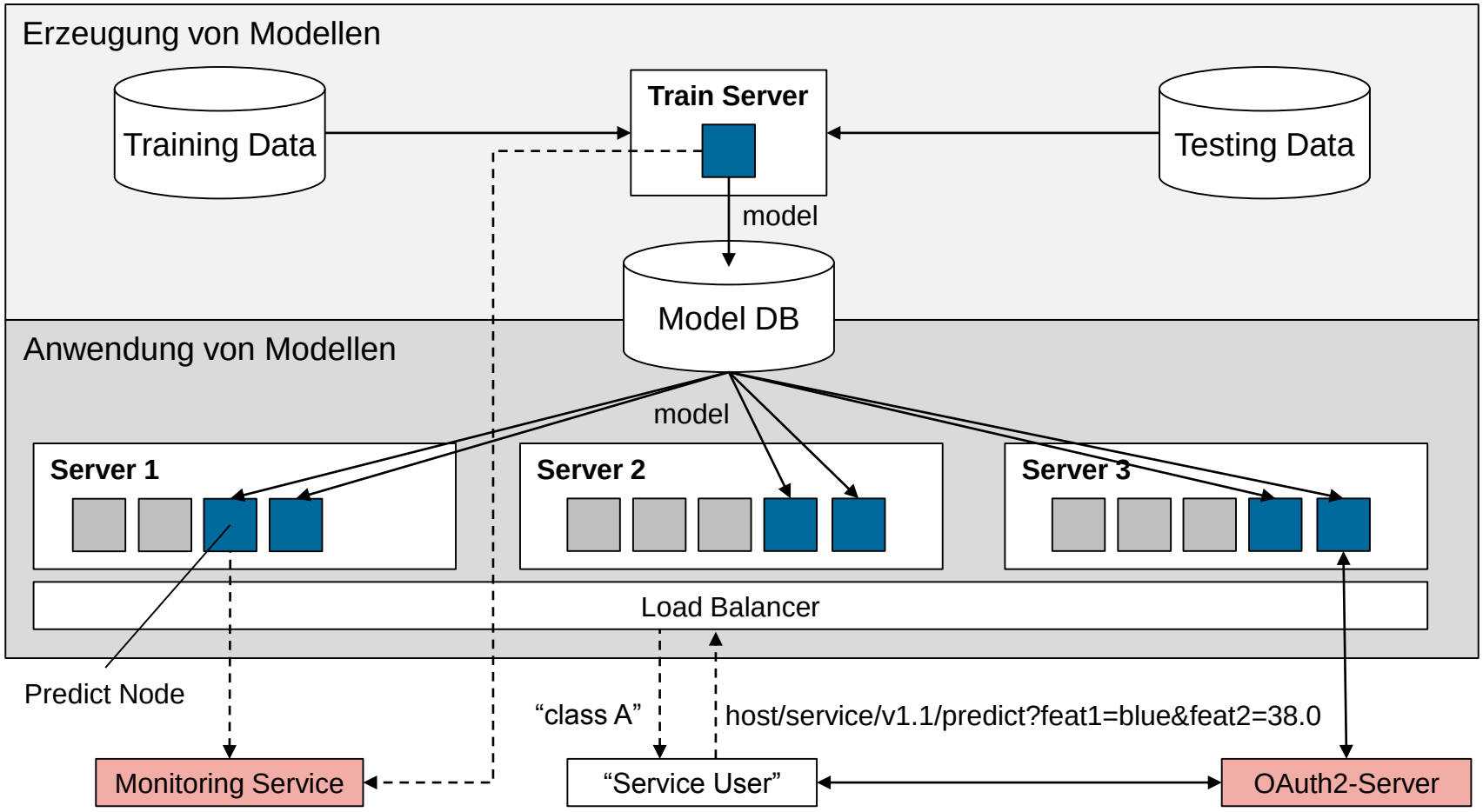
Architektur



otto group

Architektur

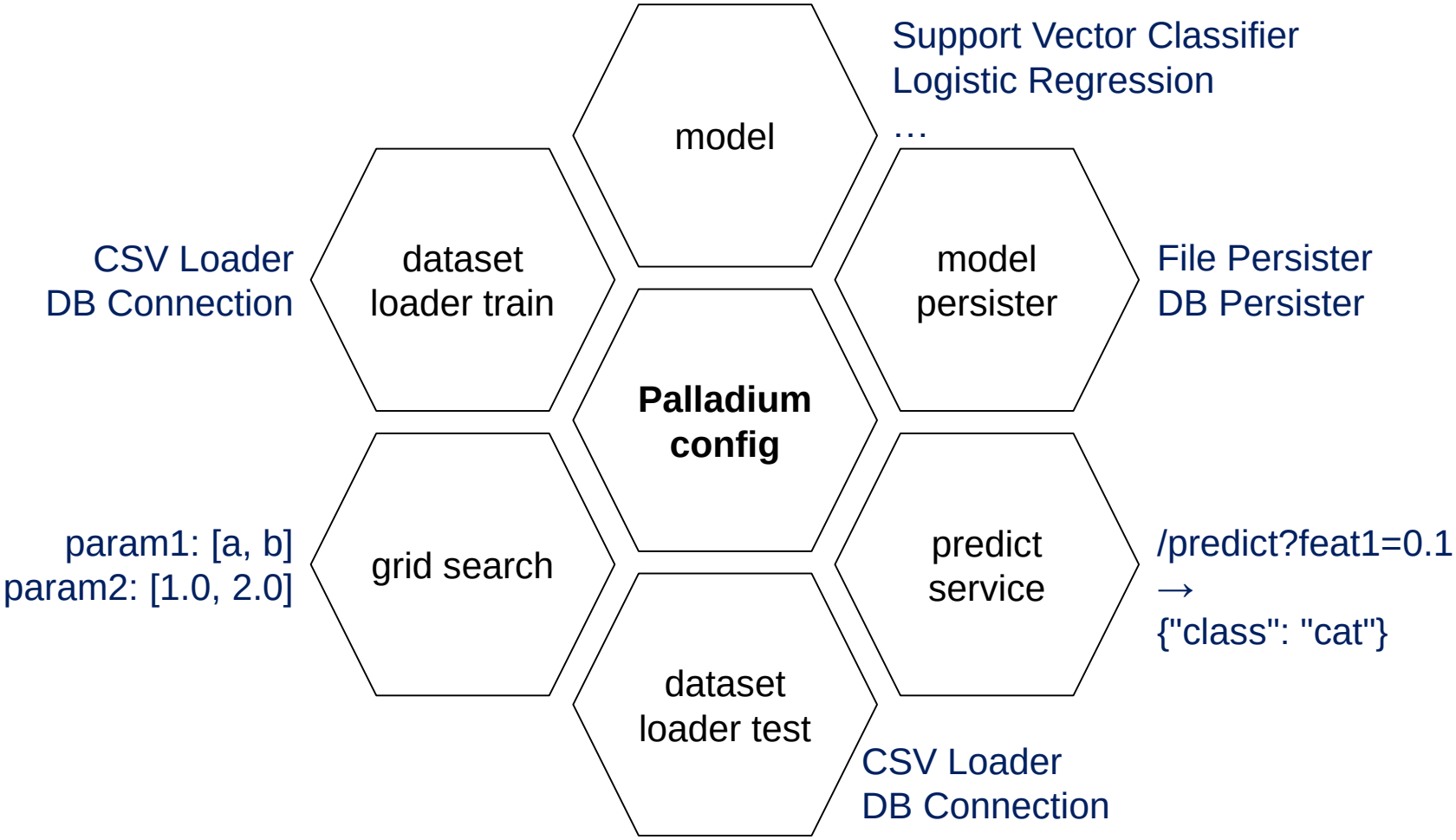
Flexibles Einbinden von Autorisierung, Logging und Monitoring



otto group

Flexible Struktur via Interfaces

Integration von scikit-learn / Nutzung des Interfaces



otto group

Agenda

1	Einleitung
2	Framework und Architektur
3	Beispiel: Aufsetzen eines Klassifikationsdienstes mit Palladium
4	Deployment mit Docker und Mesos / Marathon
5	Zusammenfassung

otto group

Beispiel: Iris-Klassifikation

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

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

?

Training- & Testdaten

```
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-setosa",
 "status":"OK"}
```

Palladium Predict Server

otto group

Konfiguration und zugehörige Klassen

```
'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

Konfiguration

```
'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,
```

Konfiguration

```
'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,
```

Konfiguration

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

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

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

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

Konfiguration

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

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

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

```
'param_grid': {
    'min_samples_leaf':
        [1, 2, 3],
},
```

Konfiguration

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

'dataset_loader_test': {...},

'model': {...},

'grid_search': {...},

'model_persister': {...},

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

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


otto group

Konfiguration

```
'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'),
    ],
```

Lernen und Evaluieren von Modellen

Kommando: pld-fit

- Lädt Trainingsdaten
- Lernt Modell
- Speichert Modell + Metadaten

```
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.
```

Lernen und Evaluieren von Modellen

Kommando: pld-grid-search

- Lädt Trainingsdaten
- Teilt Trainingsdaten in “Folds” (Cross Validation)
- Trainiert und evaluiert Modelle auf verschiedenen Folds

```
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

Lernen und Evaluieren von Modellen

Kommando: pld-test

- Lädt Testdaten
- Wendet gelerntes Modell auf Testdaten an
- Zeigt Ergebnis (z.B. 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.
```


otto group

Anwenden von Modellen

Kommandos: pld-devserver & pld-stream

- Integriertes Script zum Testen des Web Services: pld-devserver
  - Einsatz des Flask Web Servers
- Empfehlung zur Nutzung eines WSGI-Containers + Web Server, z. B. gunicorn / nginx

- Predict Server
  - Lädt Modell (via Model Persister)
  - Konfiguration regelmäßiger Modell-Updates möglich
  - Bereitstellung von Web Service Entry Points (“predict”, “alive”)

- Kommando für stdin/stdout-Verarbeitung von Prognoseanfragen: pld-stream
- Zeilenweise Verarbeitung, Rückgabe der Prognose an stdout
- In Kombination mit Hive zur Erstellung großer Anzahl an Prognosen eingesetzt

Statusinformationen über aktuellen predict-Node

Entry Point /alive, z.B. http://localhost:5000/alive


```
{
  "memory_usage": 68,
  "palladium_version": "1.0",
  "service_metadata": {
    "service_name": "iris",
    "service_version": "0.1"
  },
  "model": {
    "updated": "2015-09-28T16:33:53.316178",
    "metadata": {
      "train_timestamp": "2015-09-28T10:16:33.226367",
      "version": 2,
      "score_train": 0.94,
      "score_test": 0.96
    }
  }
}
```

Anwendung des Modells

Entry Point /predict

```
GET Request:
http://localhost:5000/predict?sepal%20length=5.2&sepal%20width=3
.5&petal%20length=1.5&petal%20width=0.2

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


otto group

Performanztest des Service-Overheads (1 CPU)

Inklusive Prognose durch Iris-Modell; Verwendung des Entwicklungs-Webservers von Flask

```
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

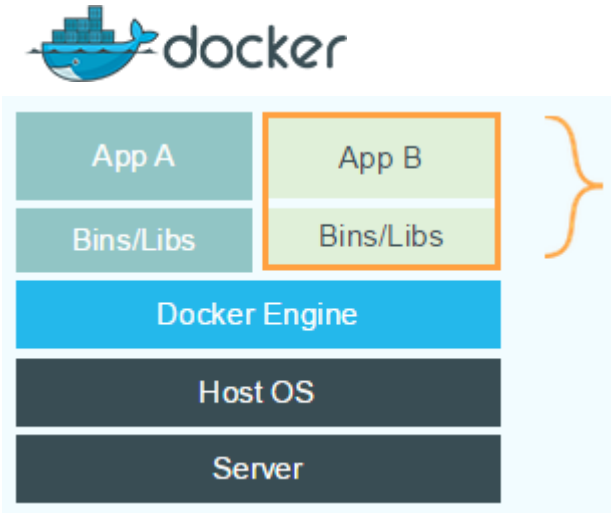
Agenda

1	Einleitung
2	Framework und Architektur
3	Beispiel: Aufsetzen eines Klassifikationsdienstes mit Palladium
4	Deployment mit Docker und Mesos / Marathon
5	Zusammenfassung

otto group

Docker, Mesos & Marathon

- Docker ist eine Plattform für die Erzeugung, Verteilung & Ausführung von Applikationen
- Leichtgewichtige Umgebung
- Einfache Kombination von Komponenten
- Gesamtpaket inkl. Abhängigkeiten
- Docker Hub für Deployment



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

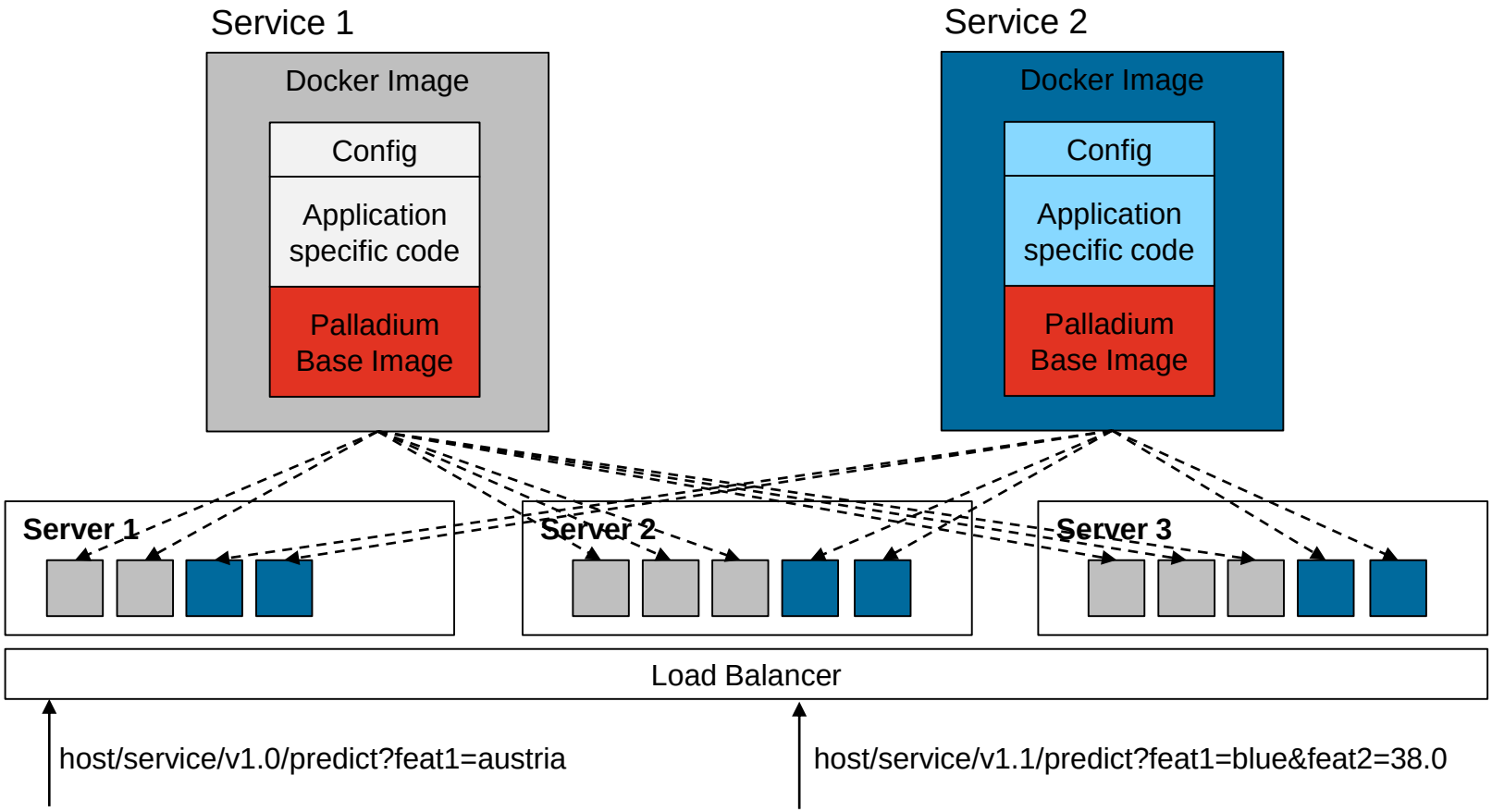
- Cluster-Framework Mesos abstrahiert von eingesetzter Hardware und stellt Ressourcen bereit
- Hohe Skalierbarkeit und Ausfallsicherheit
- Marathon (Mesosphere): Framework zum Initialisieren und Überwachen von Diensten im Cluster; Einsatz in Kombination mit Mesos

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

otto group

Deployment von Palladium-Instanzen via Docker

Skript zum automatischen Erzeugen von Docker-Images für Palladium-basierte Dienste



otto group

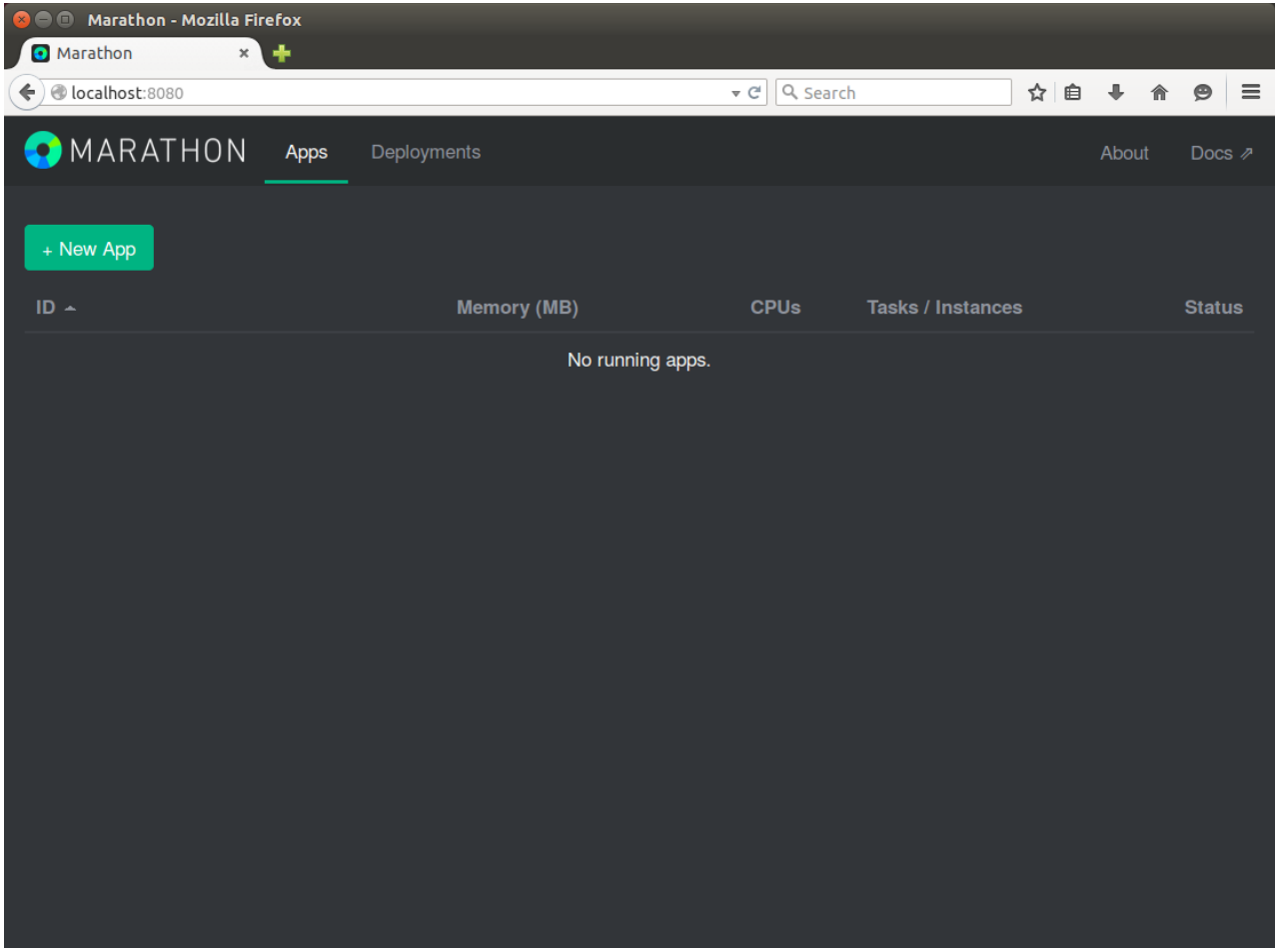
Automatisches Erzeugen von Docker-Images

Kommando: pld-dockerize

- Script pld-dockerize erzeugt Docker-Image für Prognosedienst, z.B. pld-dockerize pld_codetalks ottogroup/palladium-base:1.0 alattner/iris-demo-tmp:0.1

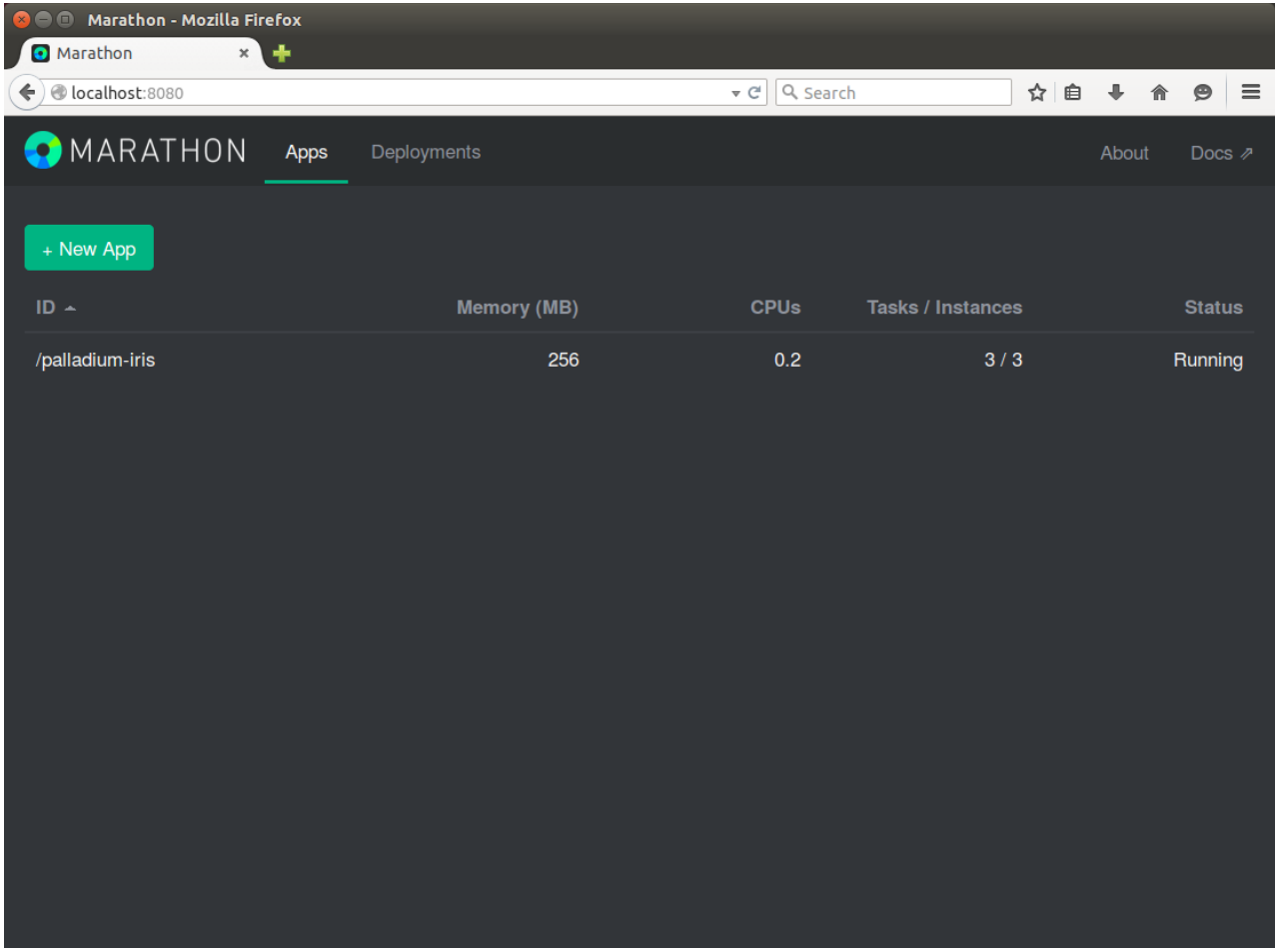
- Option, nur das Dockerfile für eine Nachbearbeitung zu erzeugen (ohne Erstellung des Images)

Deployment via Mesos / Marathon



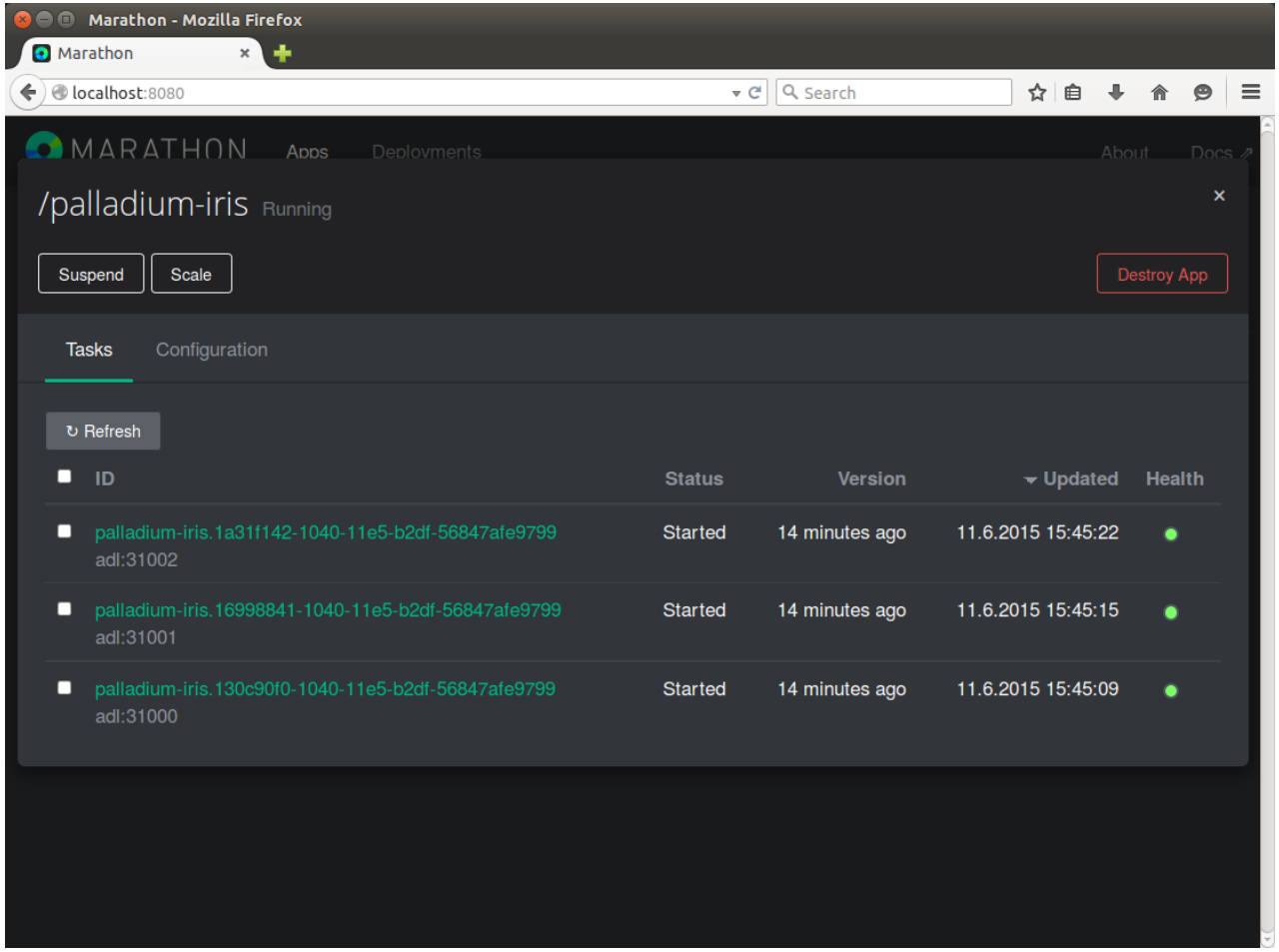
otto group

Deployment via Mesos / Marathon



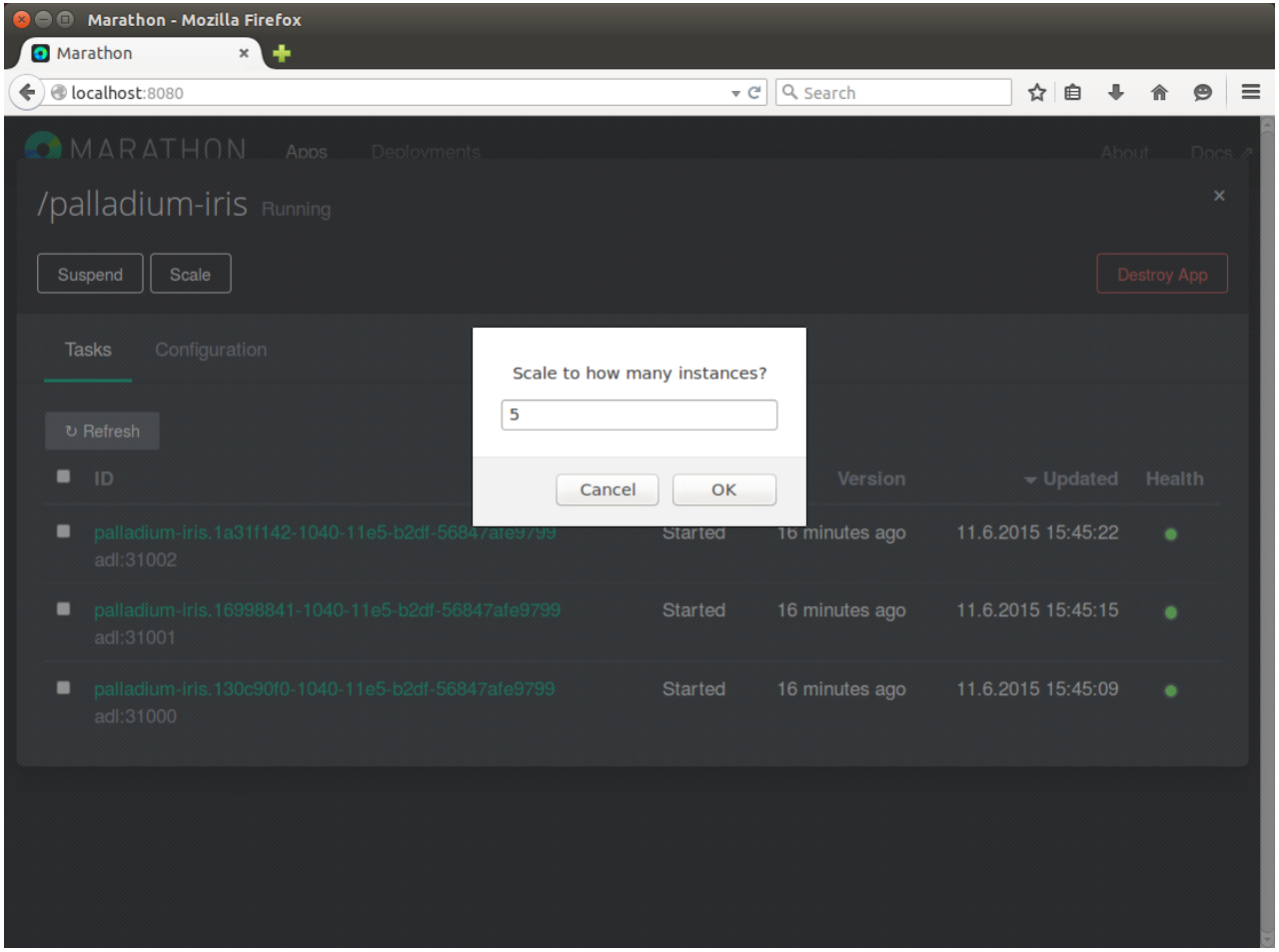
otto group

Deployment via Mesos / Marathon



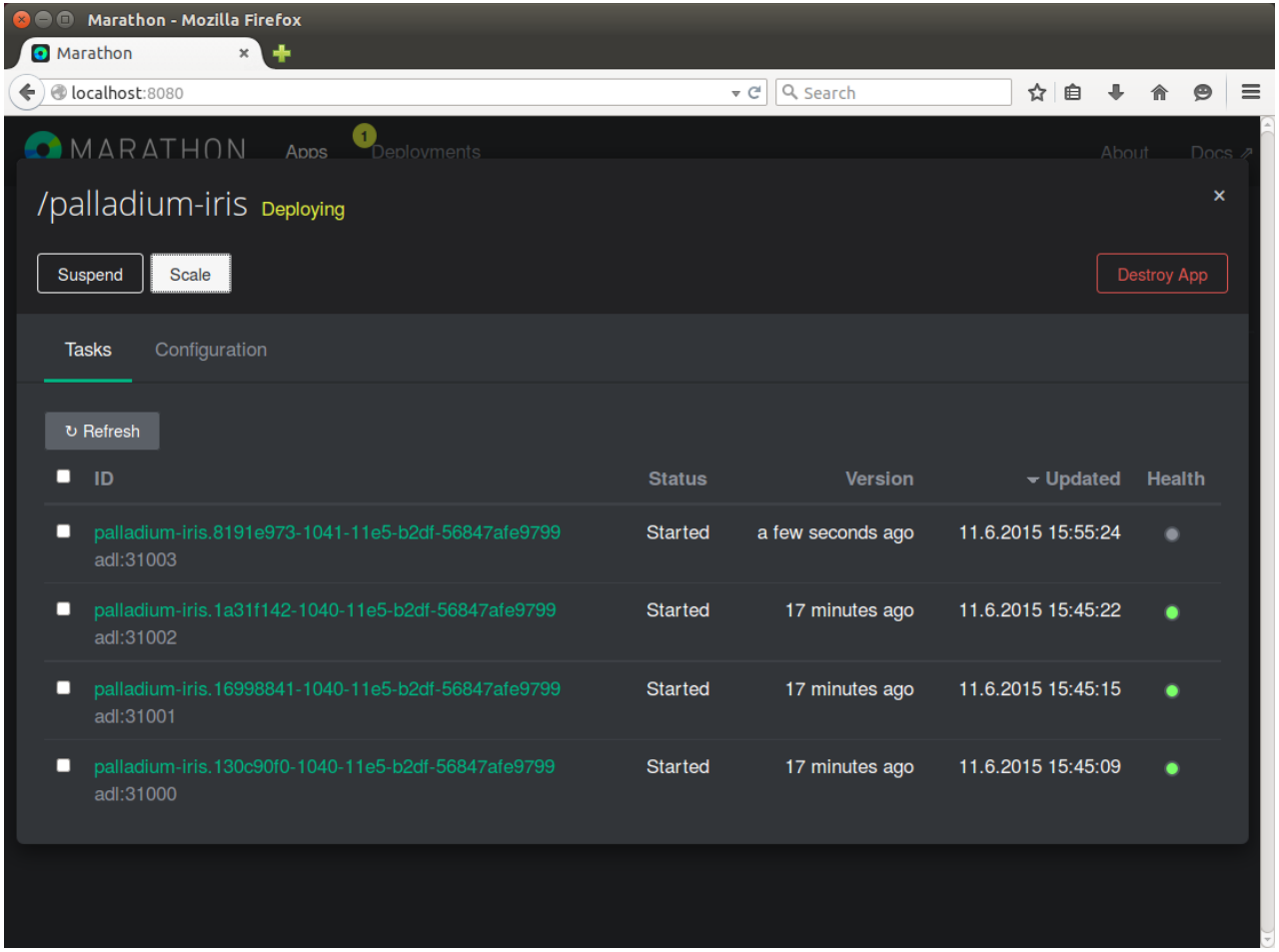
otto group

Deployment via Mesos / Marathon



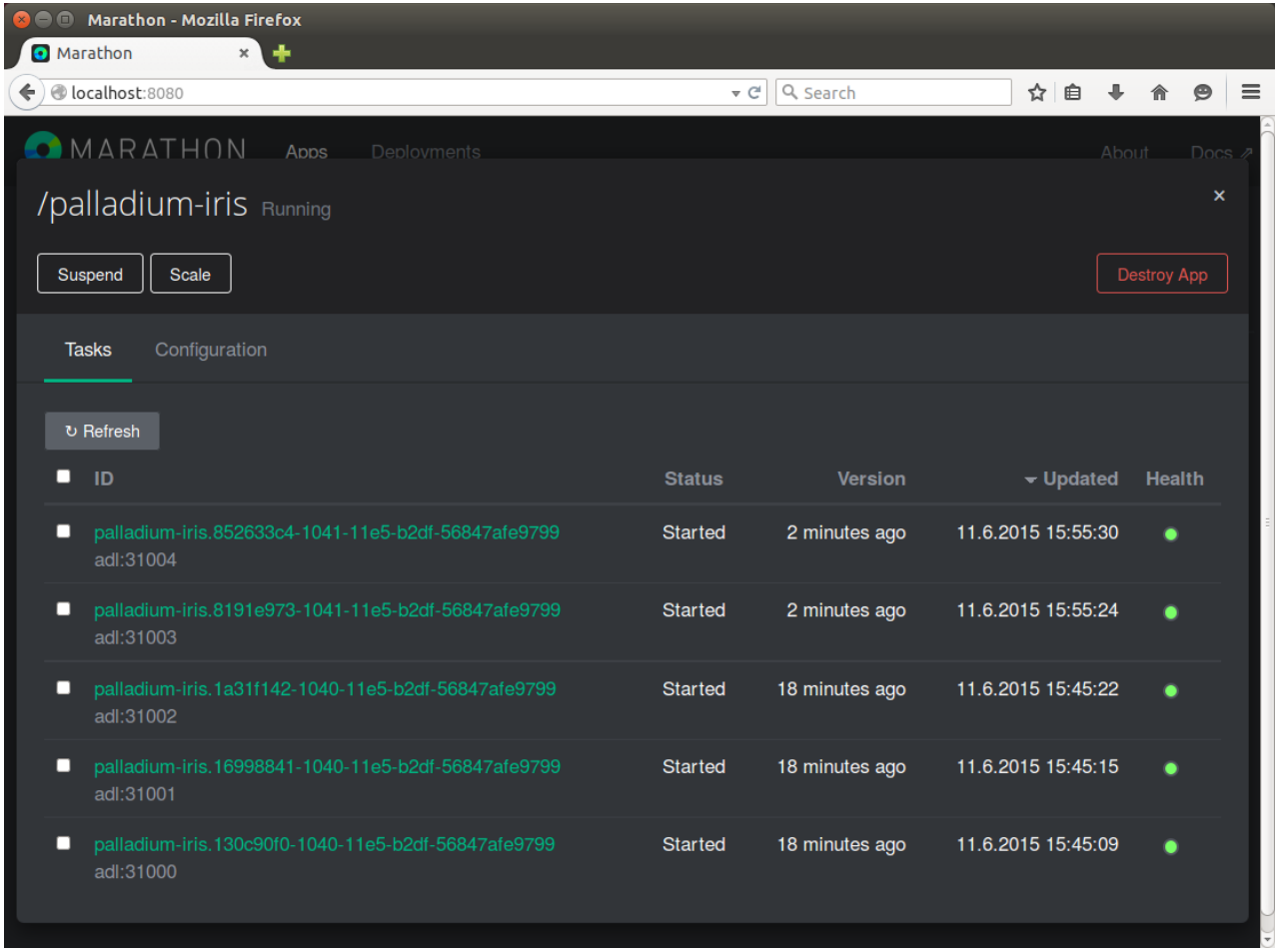
otto group

Deployment via Mesos / Marathon



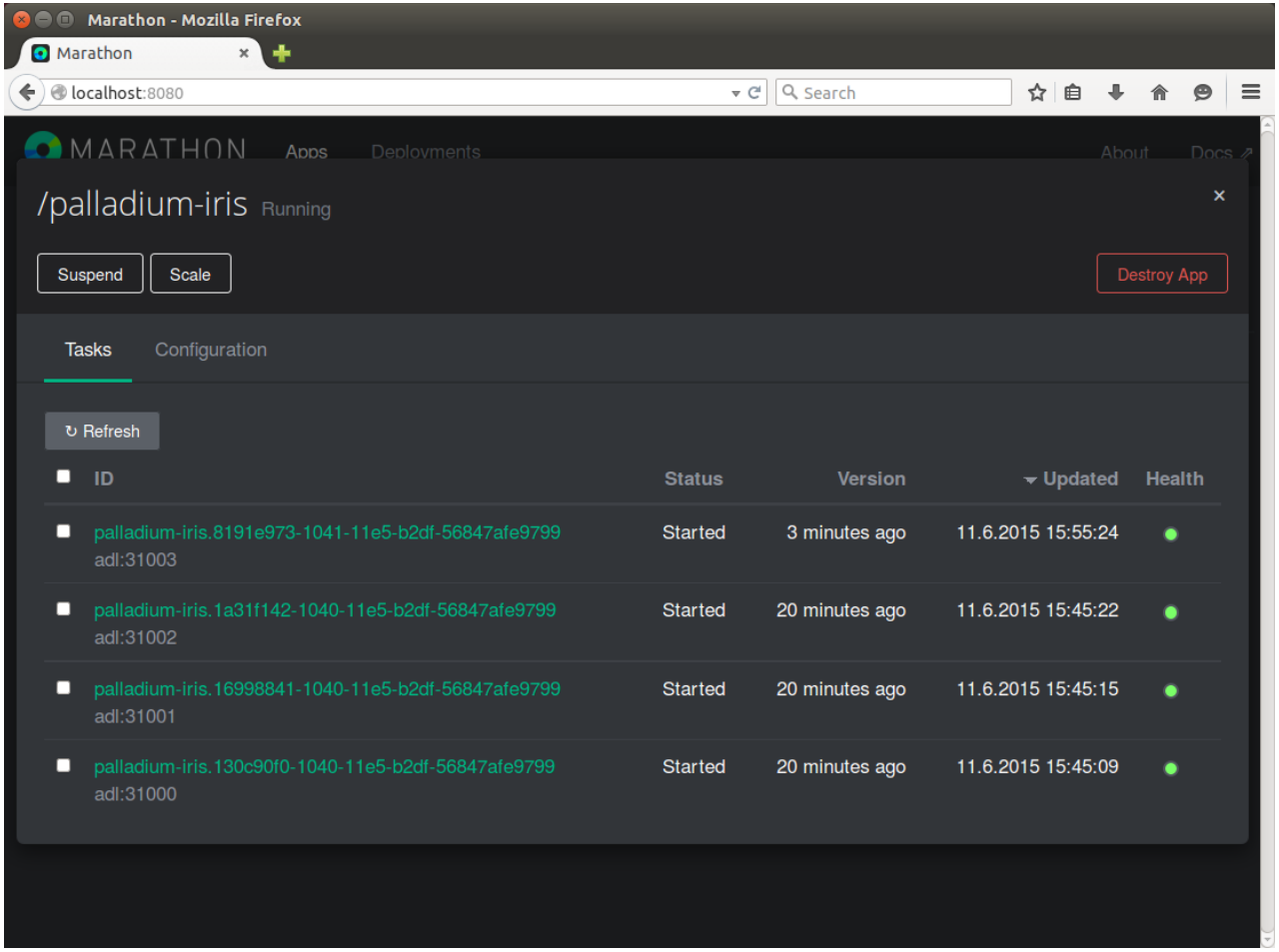
otto group

Deployment via Mesos / Marathon



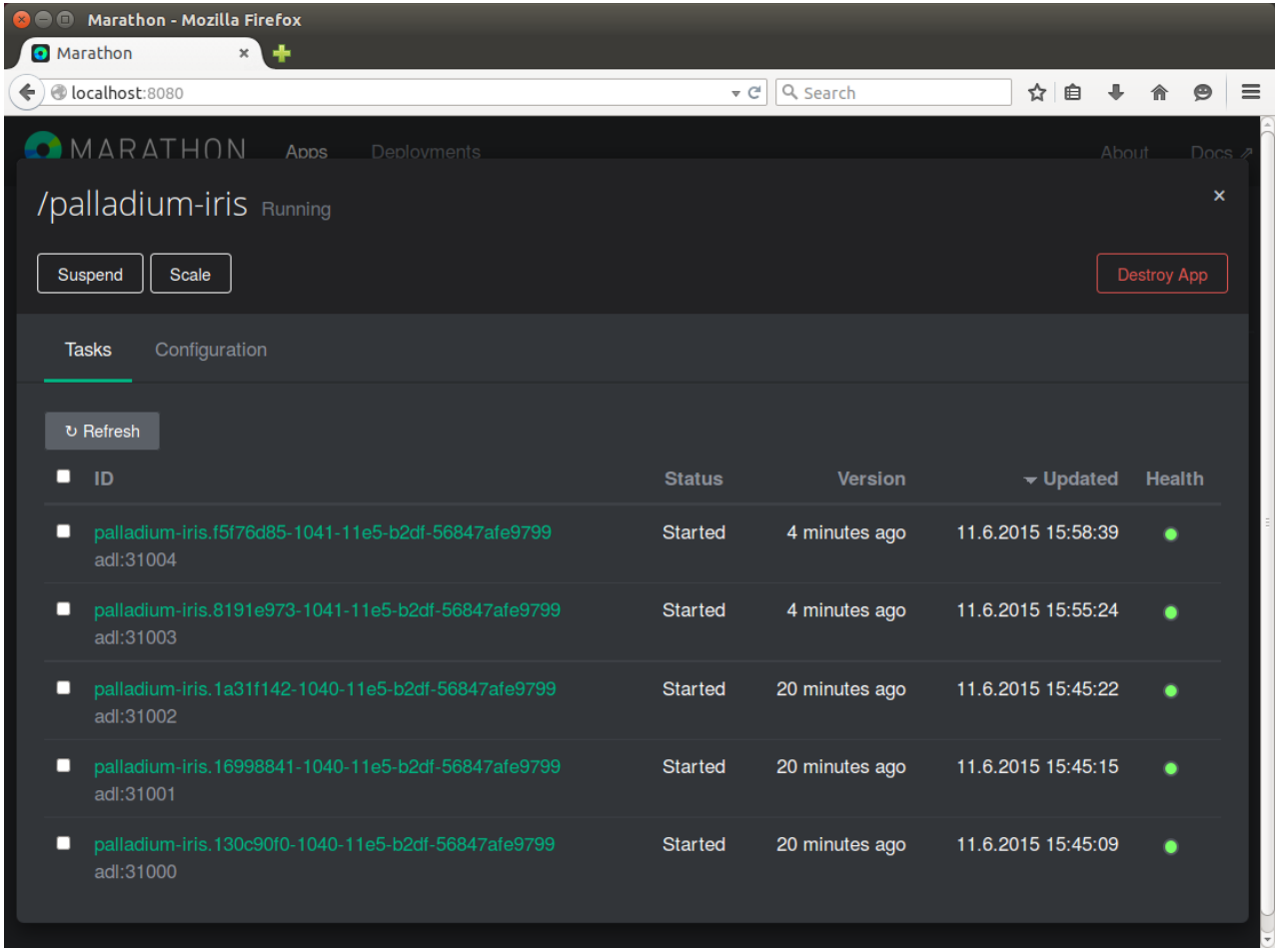
otto group

Deployment via Mesos / Marathon



otto group

Deployment via Mesos / Marathon



otto group

Deployment via Mesos / Marathon

otto group

Agenda

1	Einleitung
2	Framework und Architektur
3	Beispiel: Aufsetzen eines Klassifikationsdienstes mit Palladium
4	Deployment mit Docker und Mesos / Marathon
5	Zusammenfassung

otto group

Zusammenfassung

https://github.com/ottogroup/palladium

- Unterstützung von Modellen in Python, R und Julia
- Mechanismus für automatisches Modell-Update
- Einfache Integration von Authentifizierungs-, Logging- und Monitoringlösungen
- Administration von Modellen (Aktivieren / Löschen)
- Automatisierte Generierung von Docker-Images (pld-dockerize)
- Skalierung und Überwachung z.B. mit Mesos / Marathon

- Einfache Installation über pip oder conda
- Bereitstellung des Palladium Docker Images bei Docker Hub: https://hub.docker.com/r/ottogroup/palladium-base/
- Ausführliche Dokumentation inkl. Tutorial für Beispieldienst bei readthedocs
- Test-driven development, 100% Coverage
- Verschiedene Otto Group Services wurden mit Palladium umgesetzt

otto group

Herzlichen Dank an…

- Daniel Nouri (gemeinsame Konzeption & Entwicklung von Palladium)
- Tim Dopke (Palladium mit Docker, Mesos / Marathon)

- das Data Science-Team der Otto Group BI

- alle Entwickler der eingesetzten Pakete, u.a.
  - scikit-learn
  - numpy
  - scipy
  - pandas
  - flask
  - sqlalchemy
  - pytest
  - …

Vielen Dank für Ihre Aufmerksamkeit!

andreas.lattner //AT\\ ottogroup.com

otto group