Kubernetes for machine learning: productivity over primitives

Sophie Watson • @sophwats • sophie@redhat.com
William Benton • @willb • willb@redhat.com

Kubernetes for machine learning: productivity over primitives

Sophie Watson • @sophwats • sophie@redhat.com
William Benton • @willb • willb@redhat.com

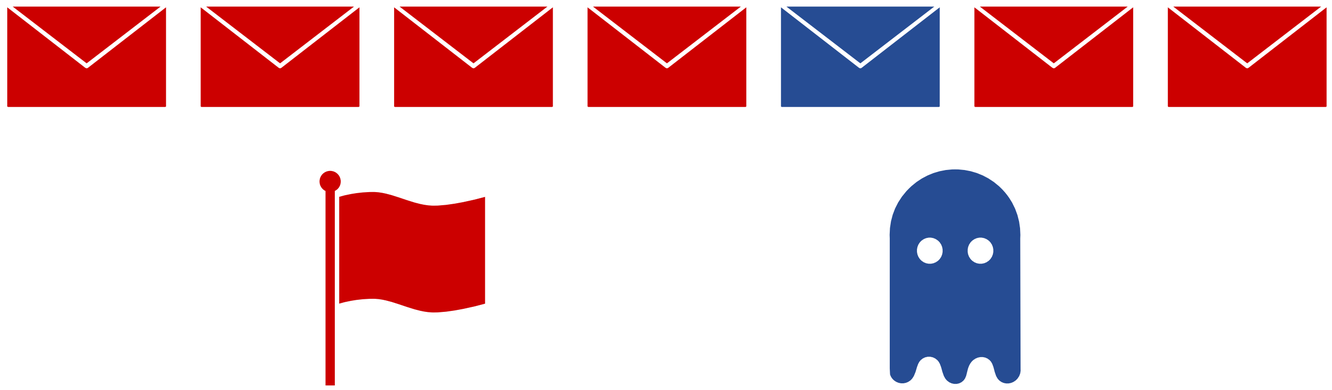
codifying problem and metrics

codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

codifying problem and metrics

codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

codifying problem and metrics



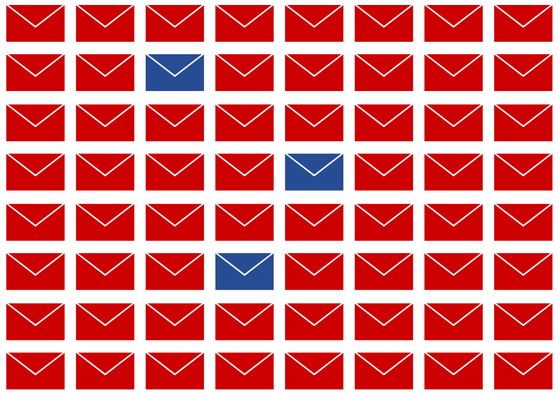
data collection and cleaning

codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

data collection and cleaning

Impedit quo id dolorum debitis qui omnis. Et ipsa animi ab ipsa blanditiis consequatur. Est consequatur cumque minima nesciunt sint. Illum rerum minus odit qui.

In quia excepturi adipisci. Maxime aut est libero atque quod. Voluptatum quae quos occaecati expedita qui impedit sunt nisi. Qui quod eligendi provident.



feature engineering

Red Hat

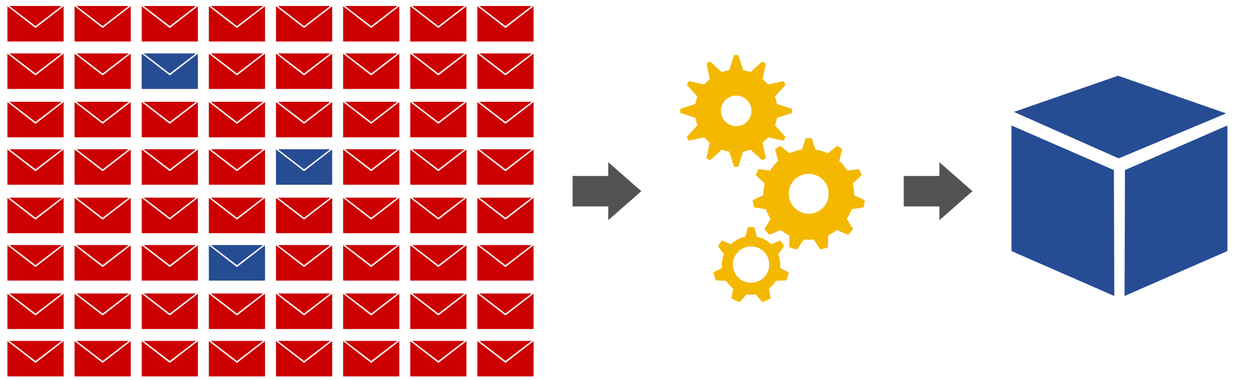
codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

feature engineering

0.67	0.57	0.84	•••	0.08	0.42	0.01

codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

model training and tuning

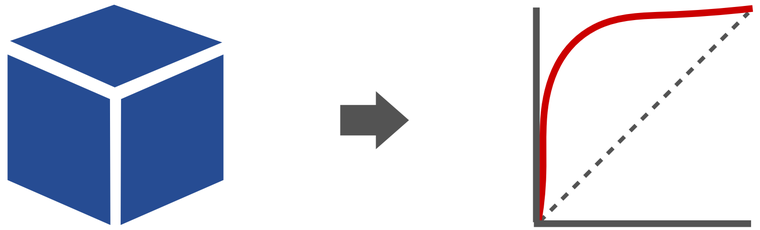


codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

model validation

codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

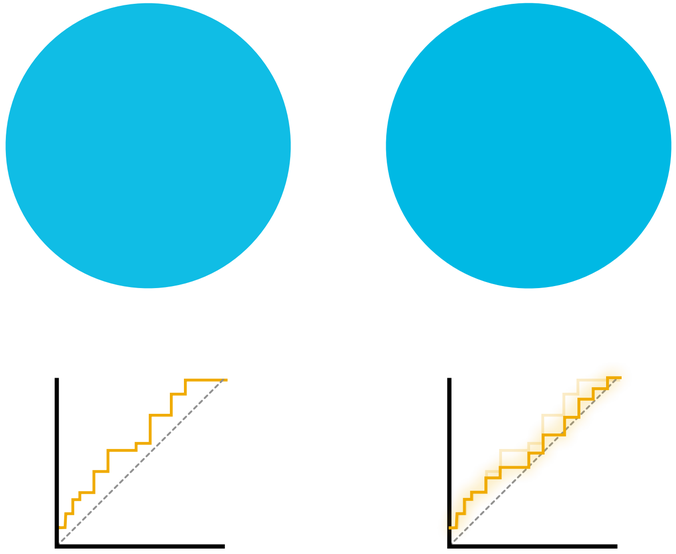
model validation



codifying problem and metrics ▶ data collection and cleaning ▶ feature engineering ▶ model training and tuning ▶ model validation ▶ model deployment ▶ monitoring, validation

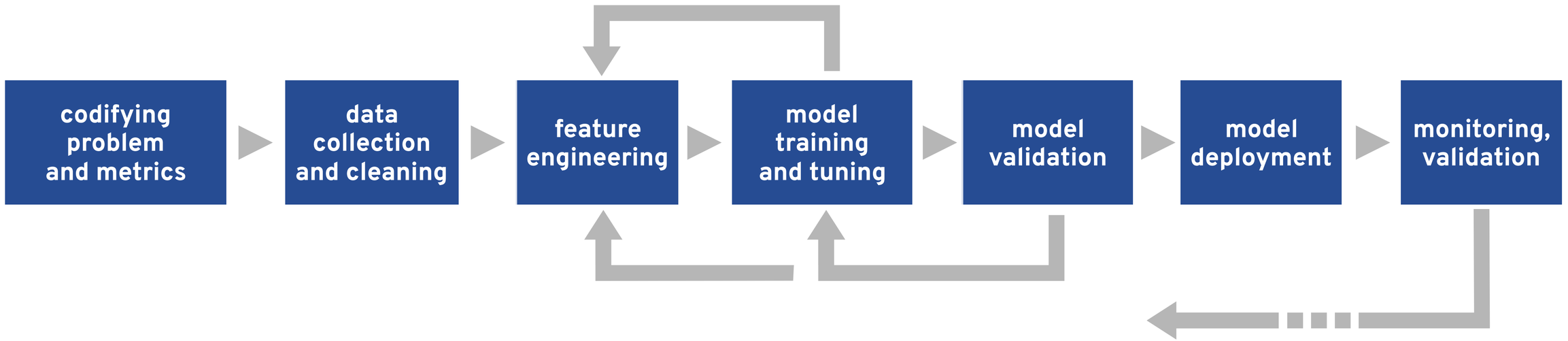
model deployment

monitoring, validation

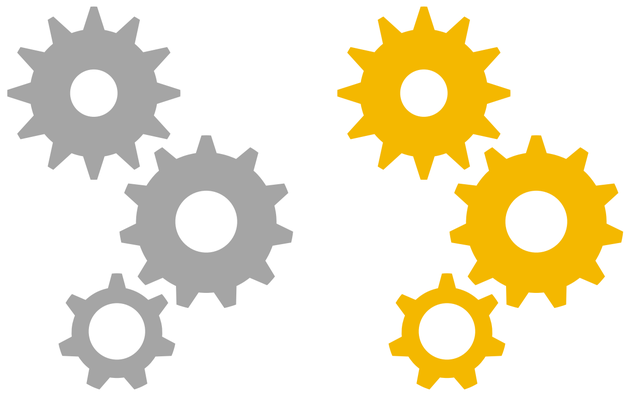


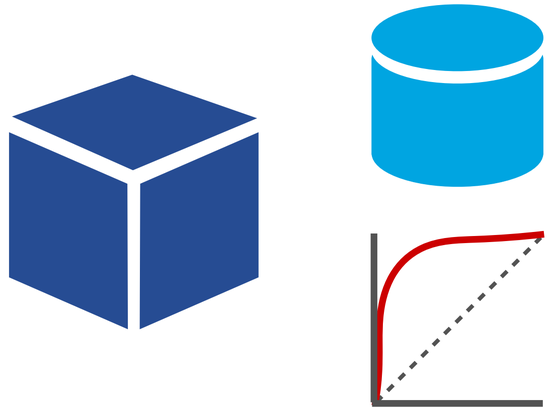
Red Hat

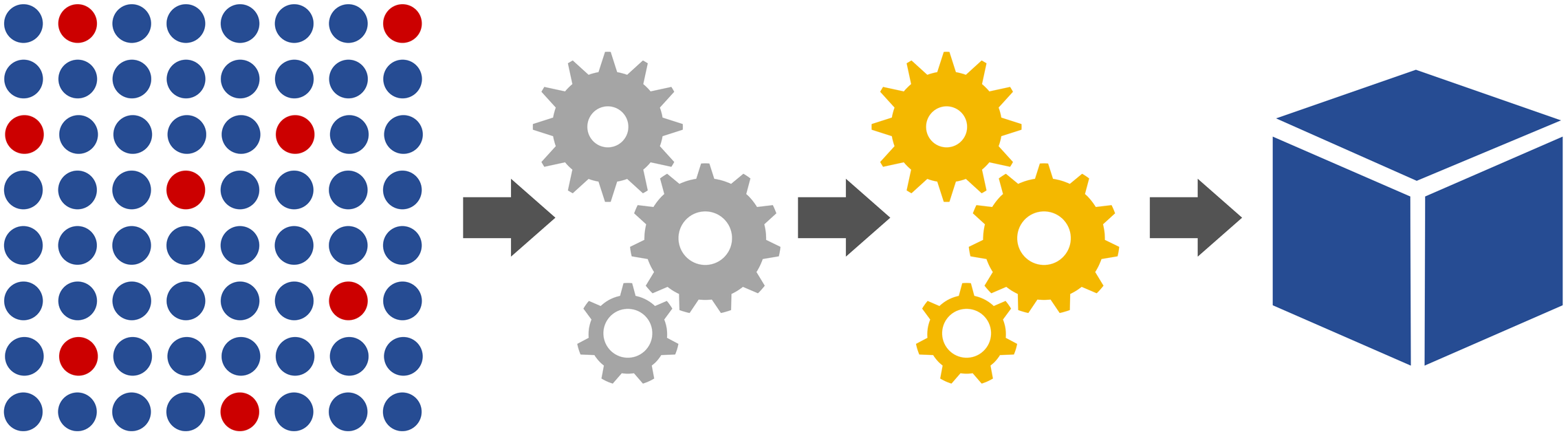

codifying problem and metrics
data collection and cleaning
feature engineering
model training and tuning
model validation
model deployment
monitoring, validation

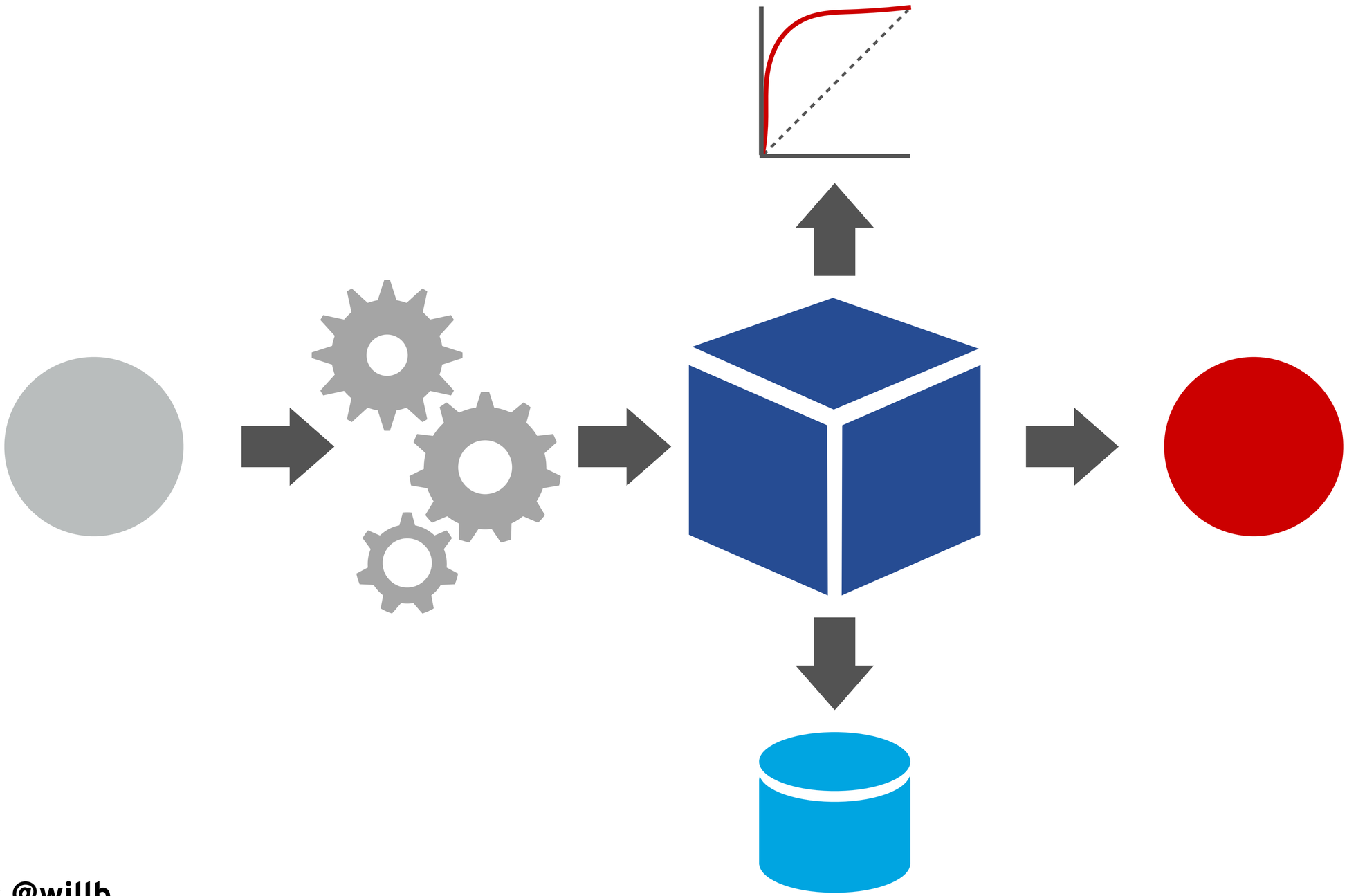

codifying problem and metrics
data collection and cleaning
feature engineering
model training and tuning
model validation
model deployment
monitoring, validation

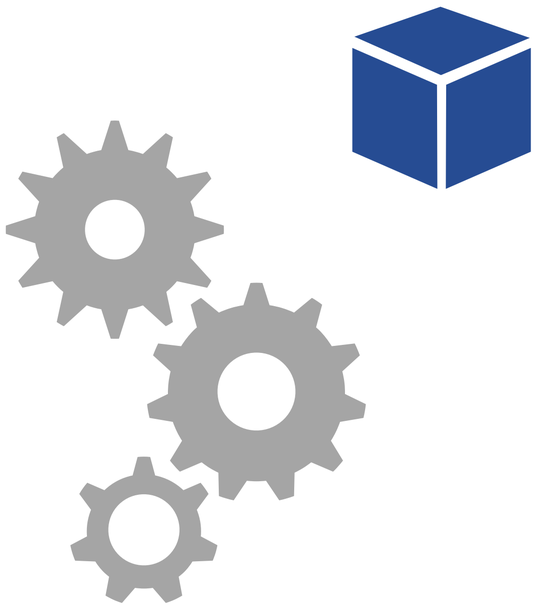
codifying problem and metrics ▶ data collection and cleaning ▶ **feature engineering** ▶ **model training and tuning** ▶ model validation ▶ **model deployment** ▶ **monitoring, validation**

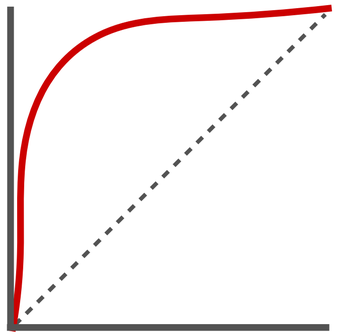


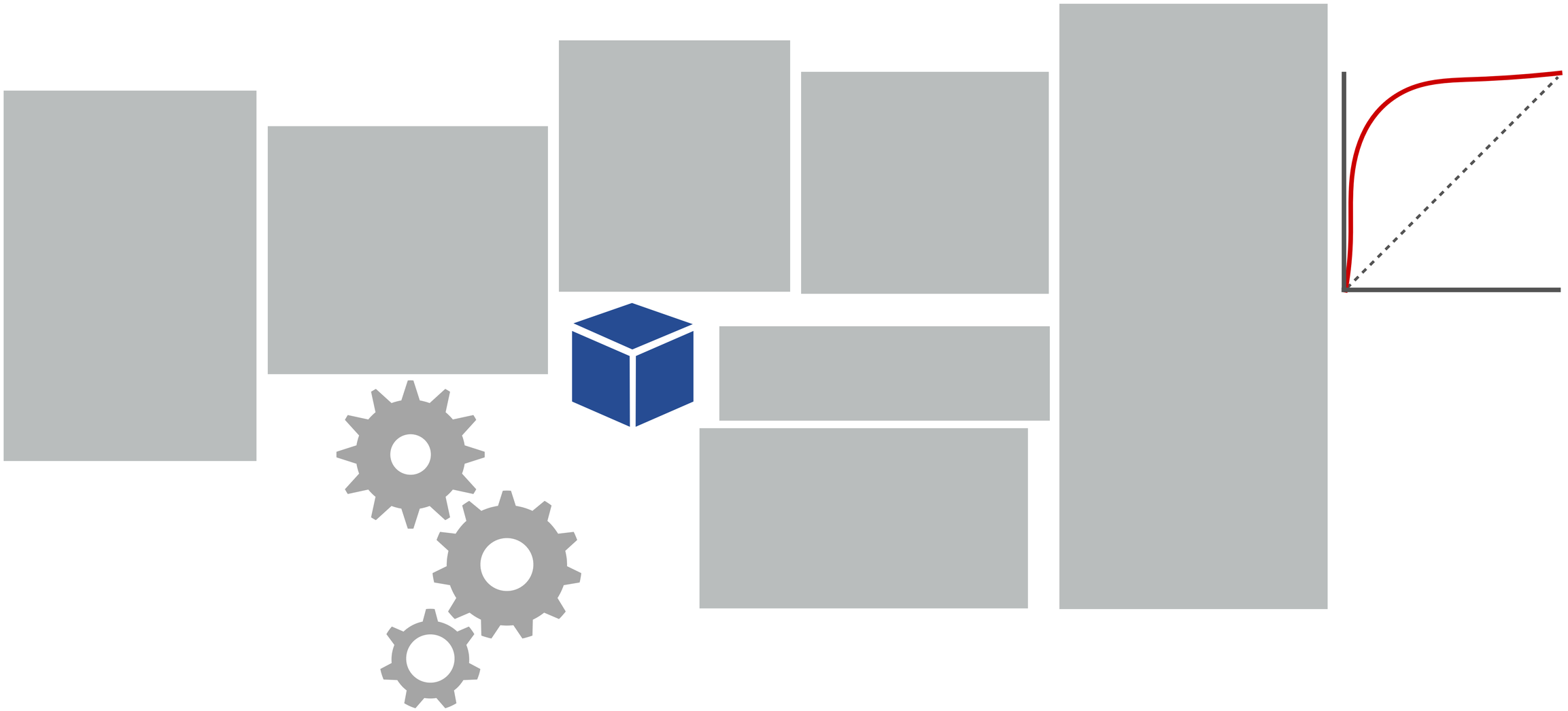


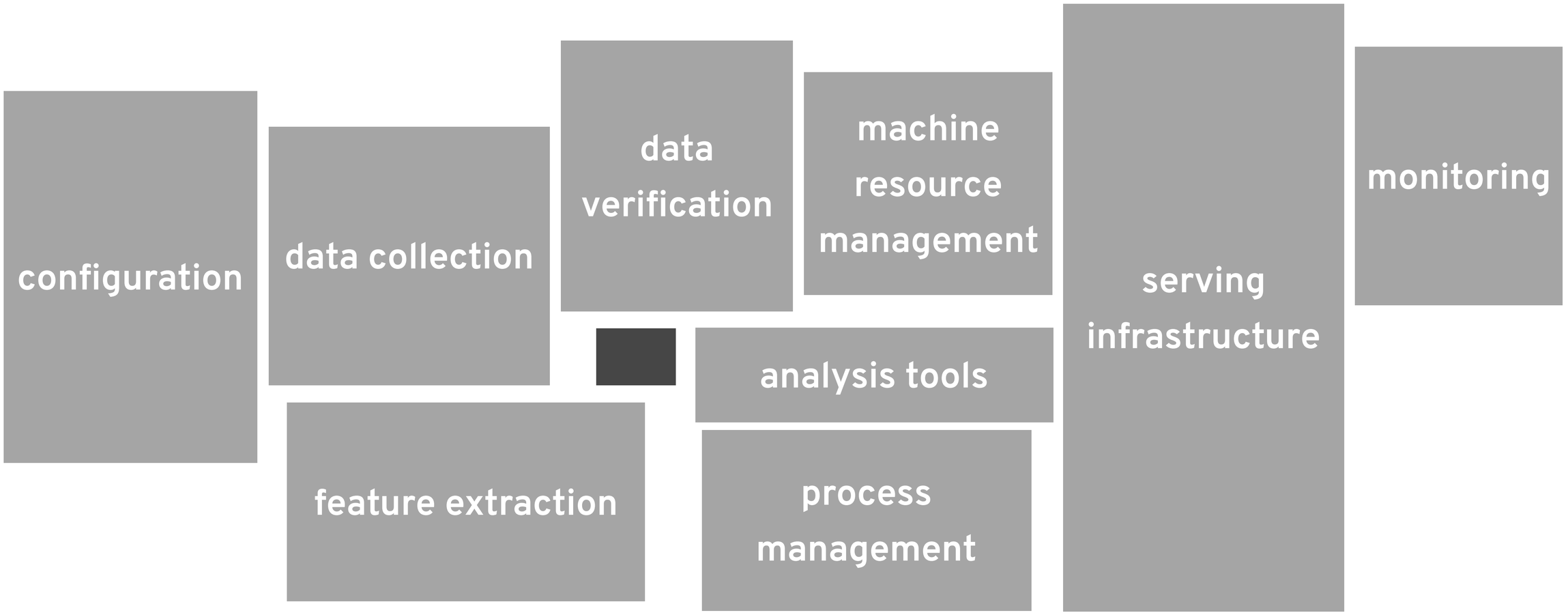




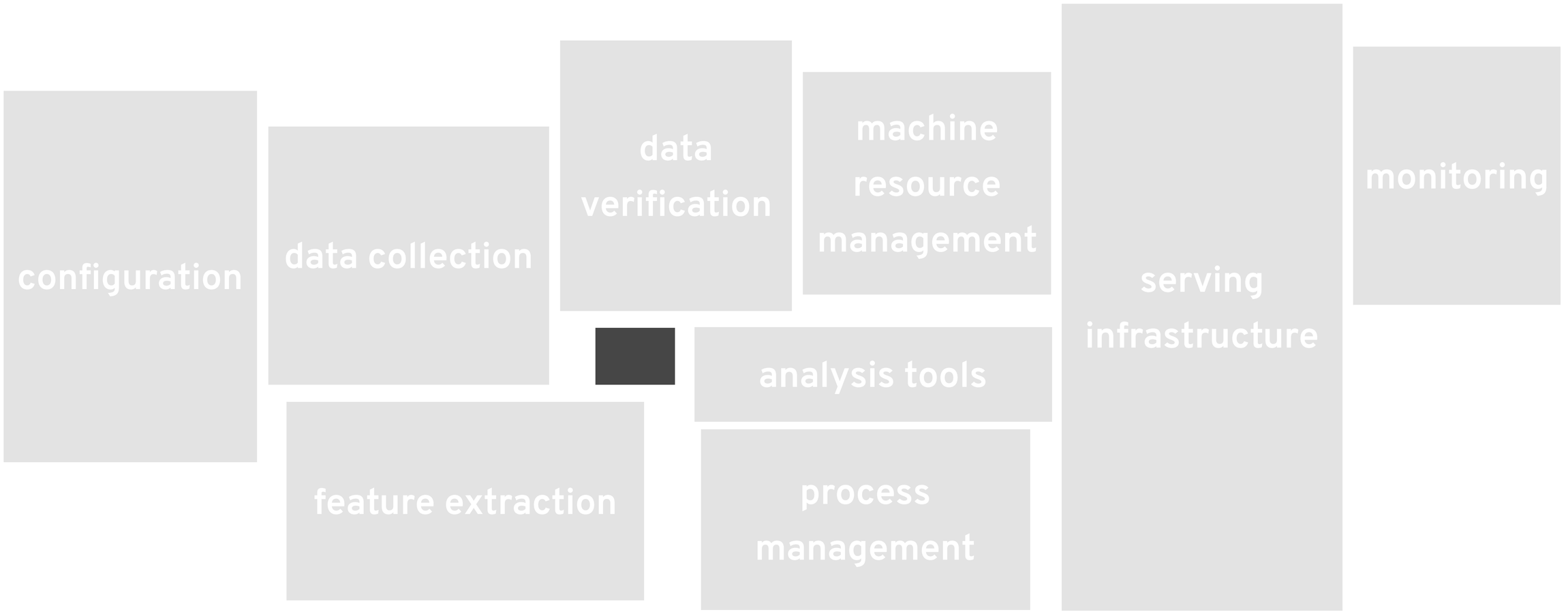




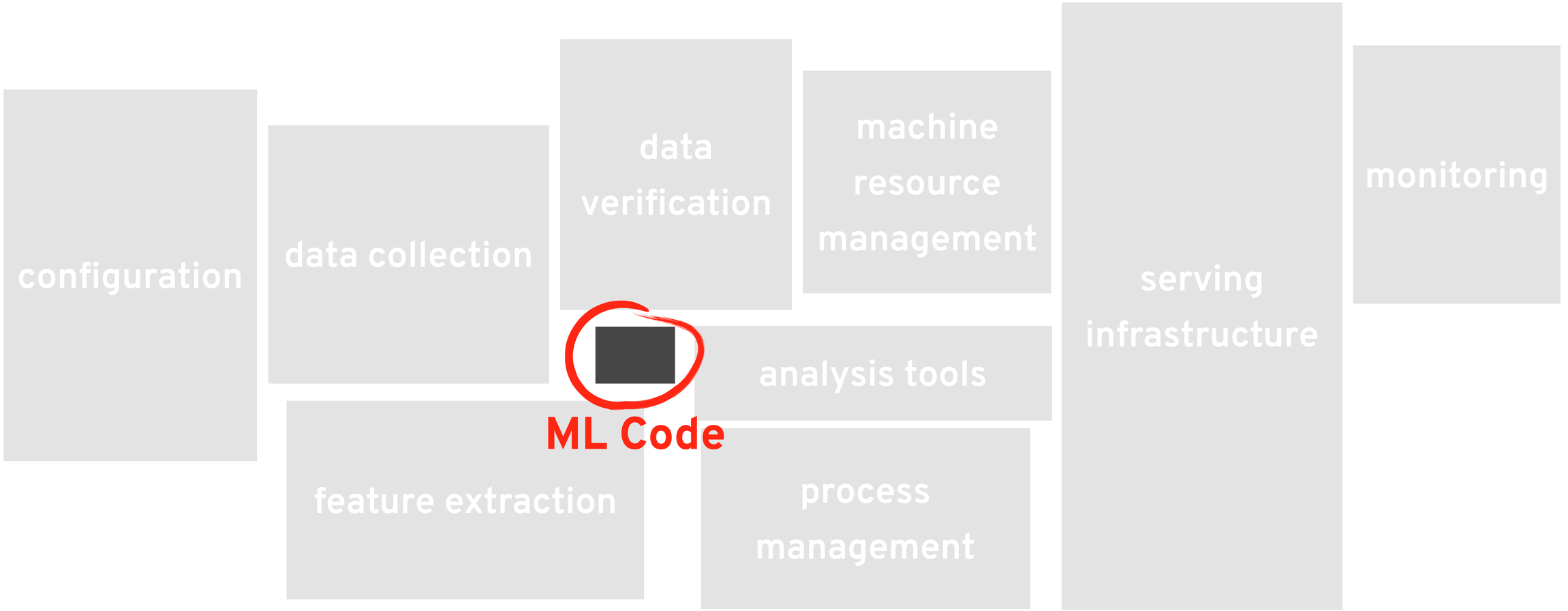




(Adapted from Sculley et al., “Hidden Technical Debt in Machine Learning Systems.” NIPS 2015)



(Adapted from Sculley et al., “Hidden Technical Debt in Machine Learning Systems.” NIPS 2015)

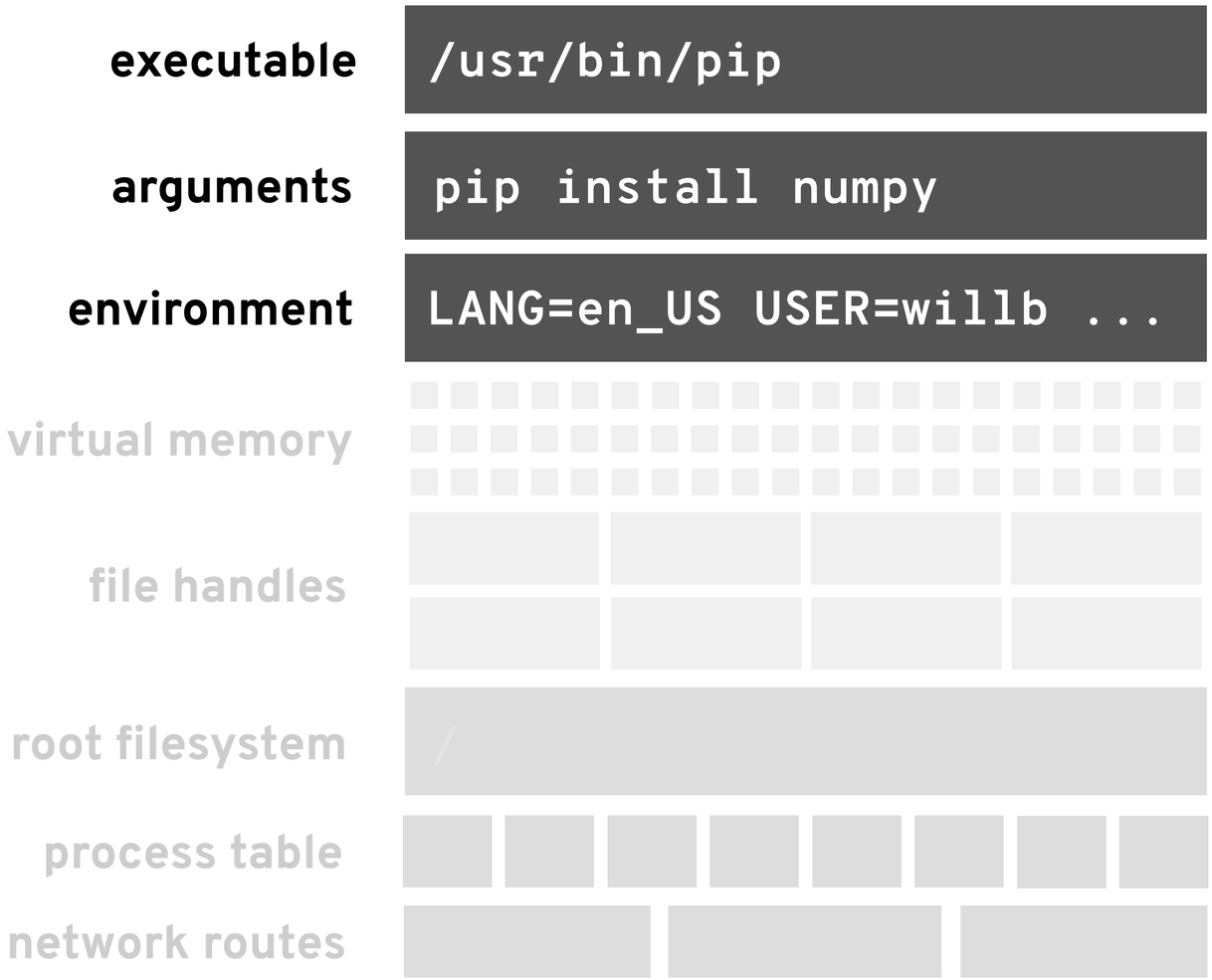


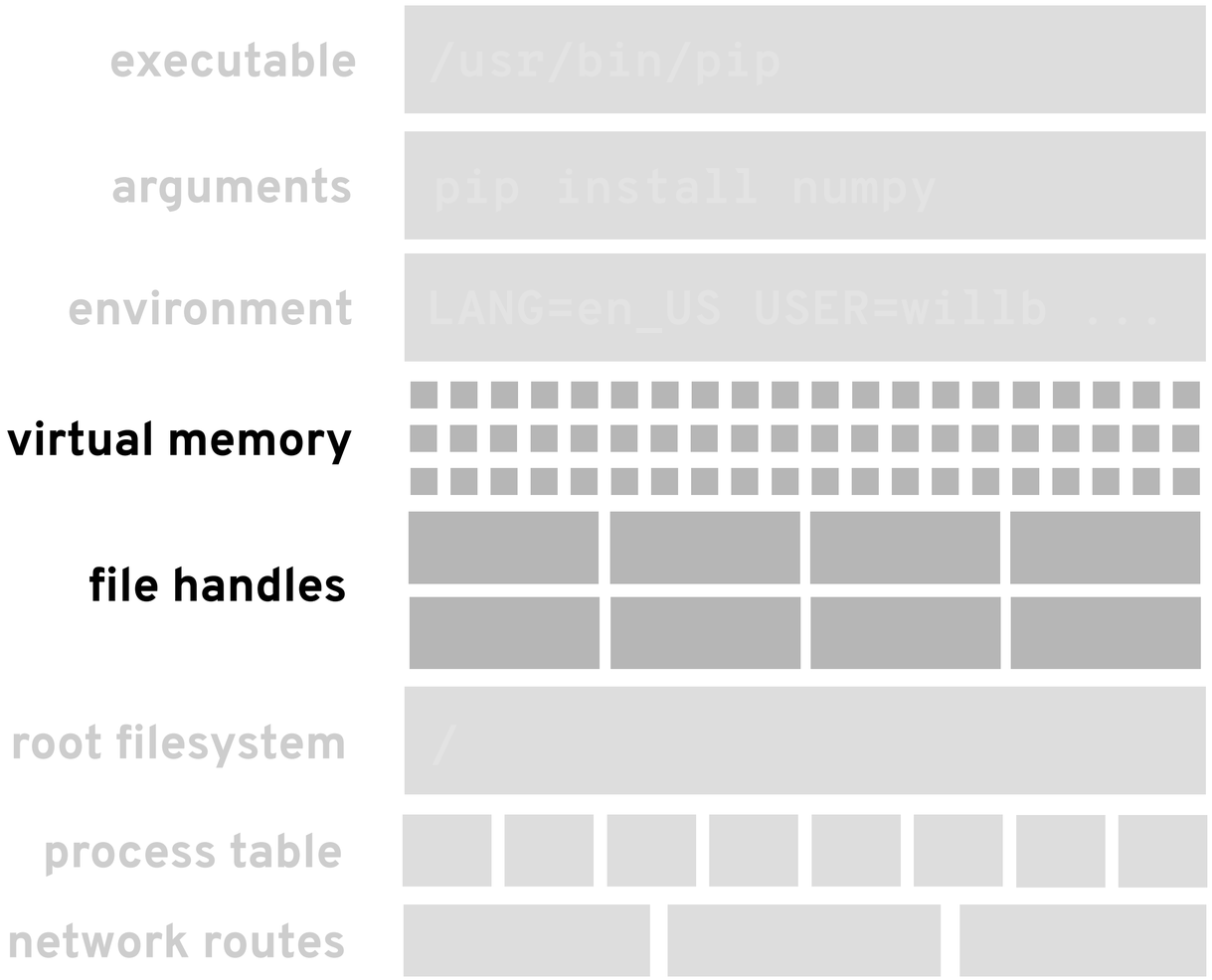
(Adapted from Sculley et al., “Hidden Technical Debt in Machine Learning Systems.” NIPS 2015)

What’s a container?

Red Hat

```
%pip install numpy
```


executable
/usr/bin/pip
arguments
pip install numpy
environment
LANG=en_US USER=willb ...
virtual memory
file handles
root filesystem
/
process table
network routes


executable
/usr/bin/pip
arguments
pip install numpy
environment
LANG=en_US USER=willb ...
virtual memory
file handles
root filesystem
/
process table
network routes

Red Hat

executable /usr/bin/pip

arguments pip install numpy

root filesystem /

process table

network routes

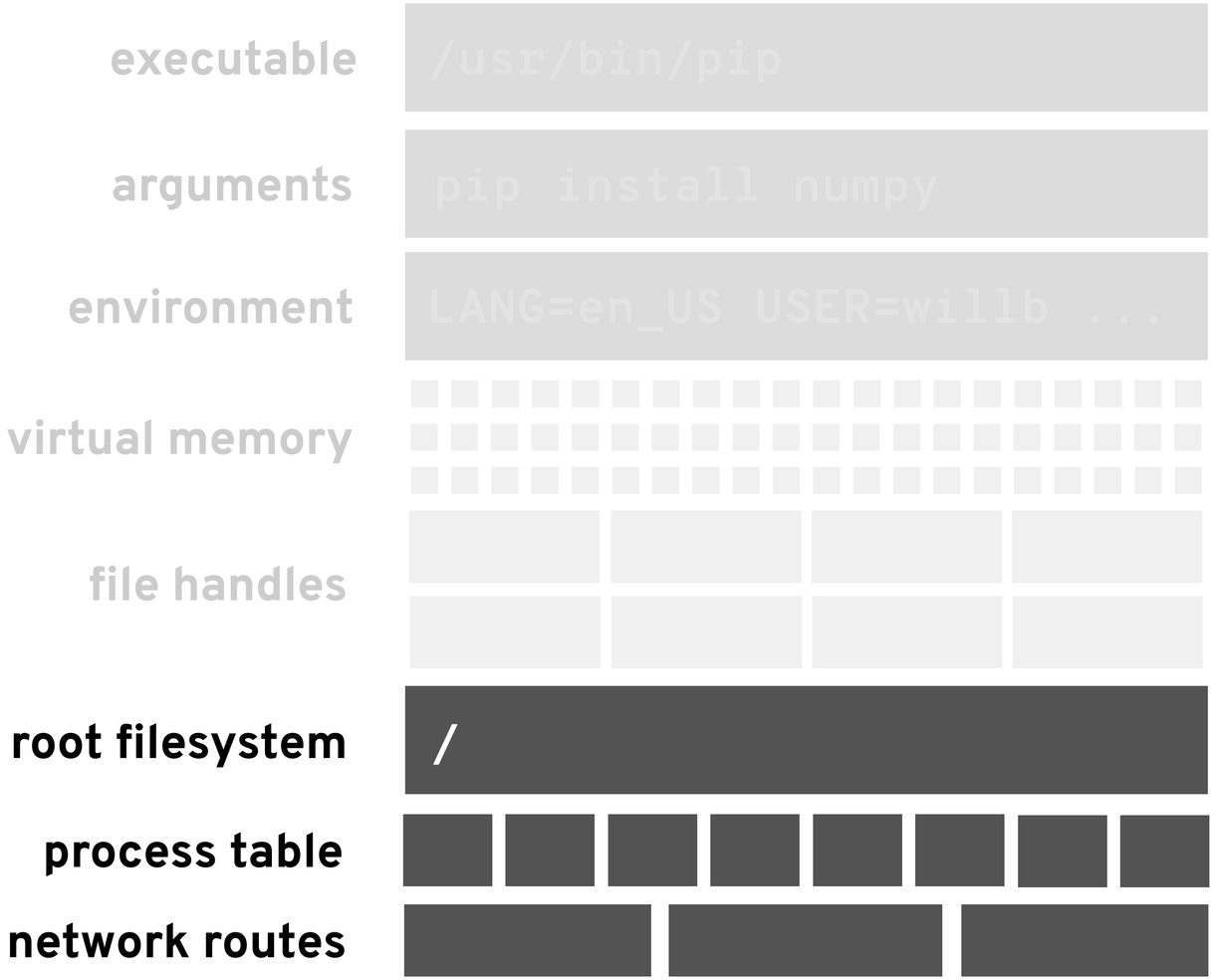
executable /usr/bin/pip

arguments pip install numpy

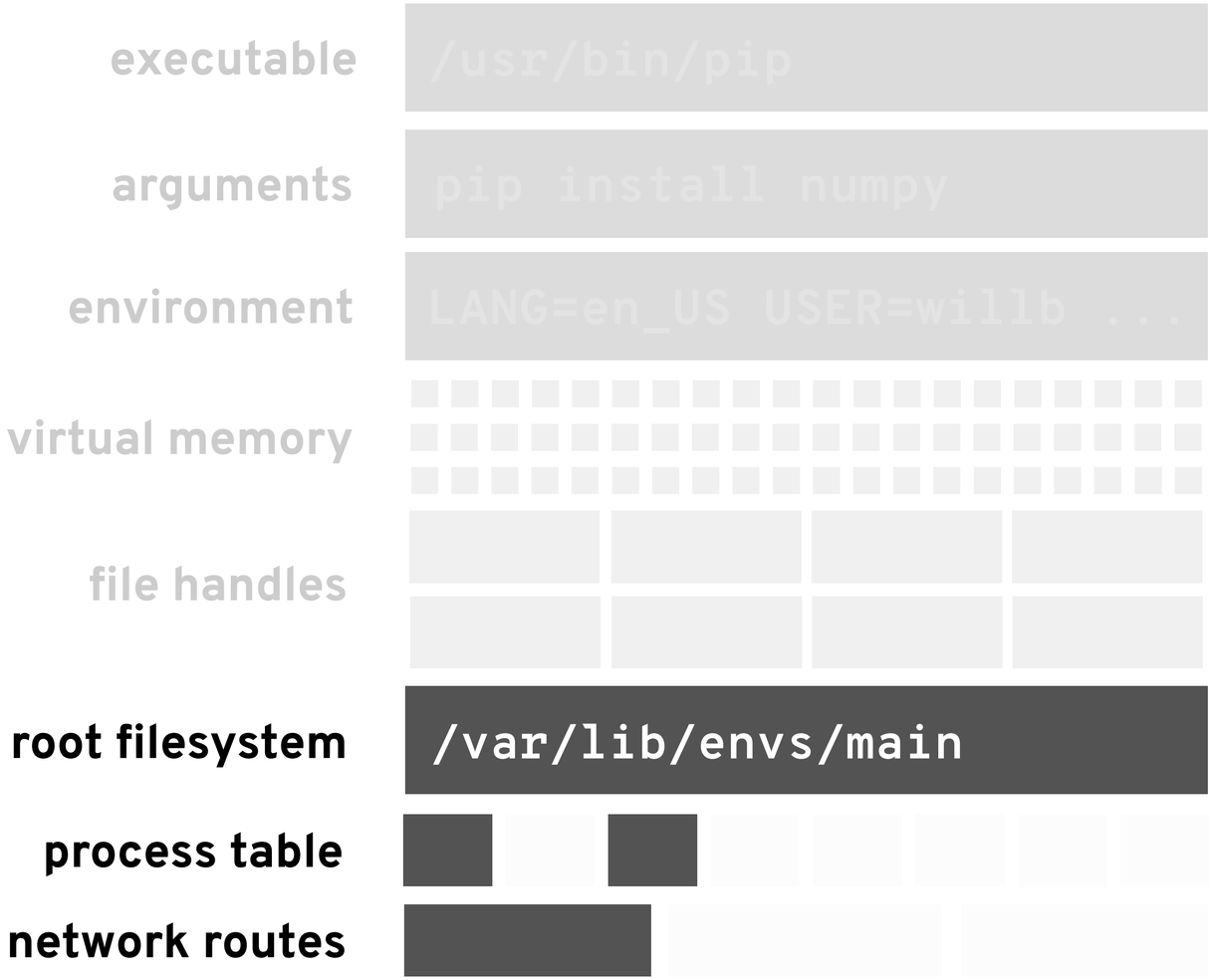
root filesystem /

process table

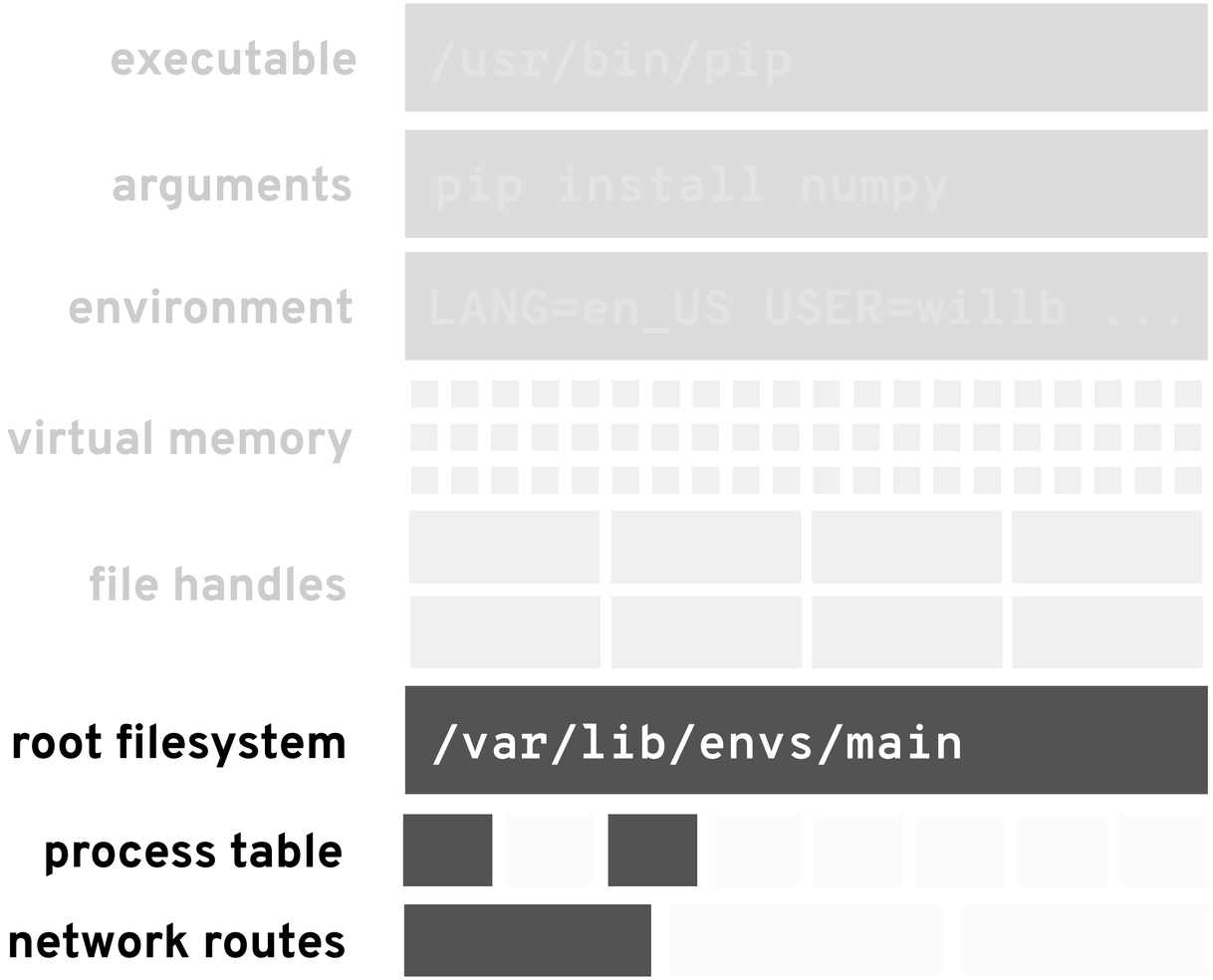
network routes


executable
/usr/bin/pip
arguments
pip install numpy
environment
LANG=en_US USER=willb ...
virtual memory
file handles
root filesystem
/
process table
network routes

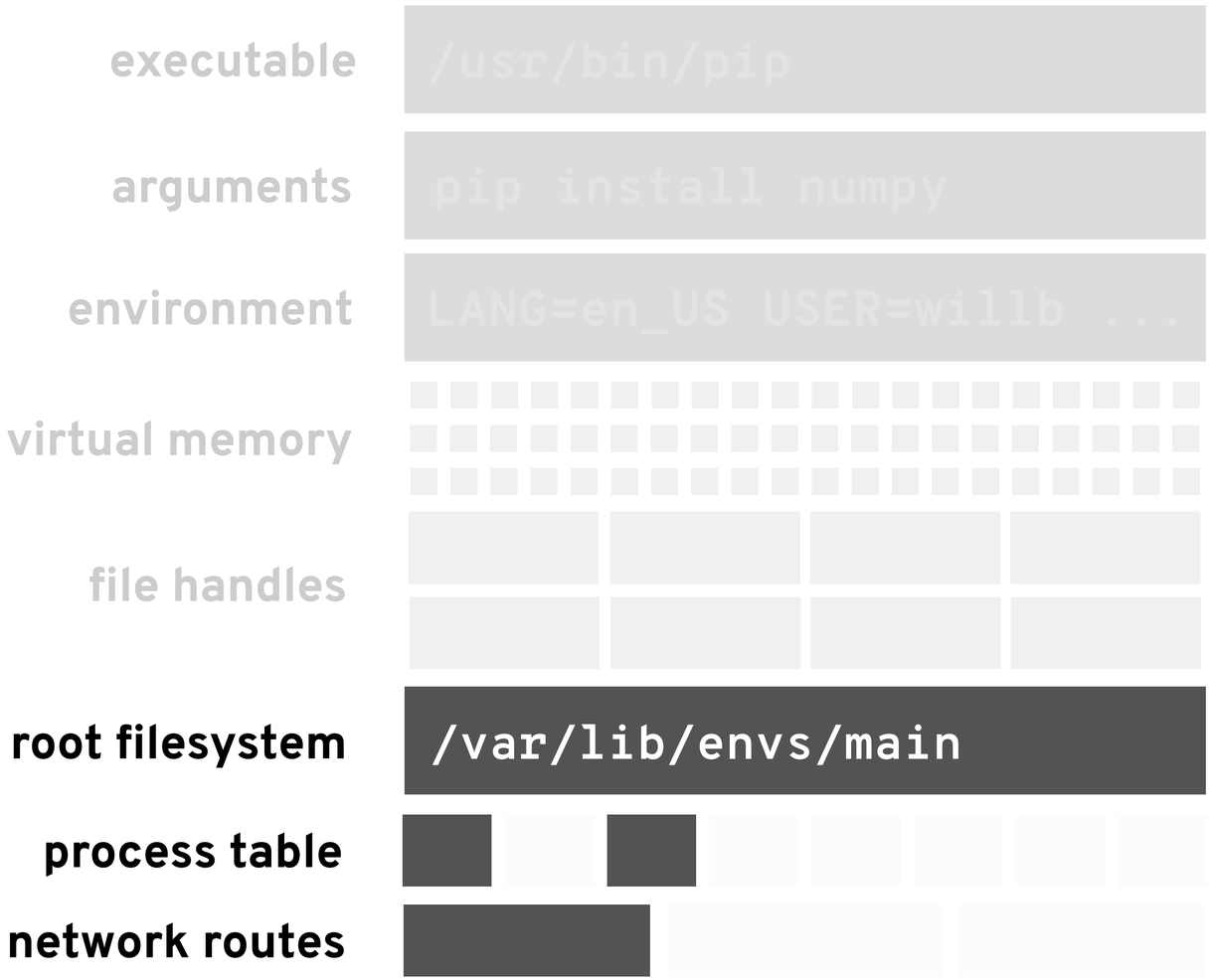
Red Hat


executable
/usr/bin/pip
arguments
pip install numpy
environment
LANG=en_US USER=willb ...
virtual memory
file handles
root filesystem
/var/lib/envs/main
process table
network routes

Red Hat


executable
/usr/bin/pip
arguments
pip install numpy
environment
LANG=en_US USER=willb ...
virtual memory
file handles
root filesystem
/var/lib/envs/main
process table
network routes

Red Hat


executable
/usr/bin/pip
arguments
pip install numpy
environment
LANG=en_US USER=willb ...
virtual memory
file handles
root filesystem
/var/lib/envs/main
process table
network routes


SPEED
LIMIT
55

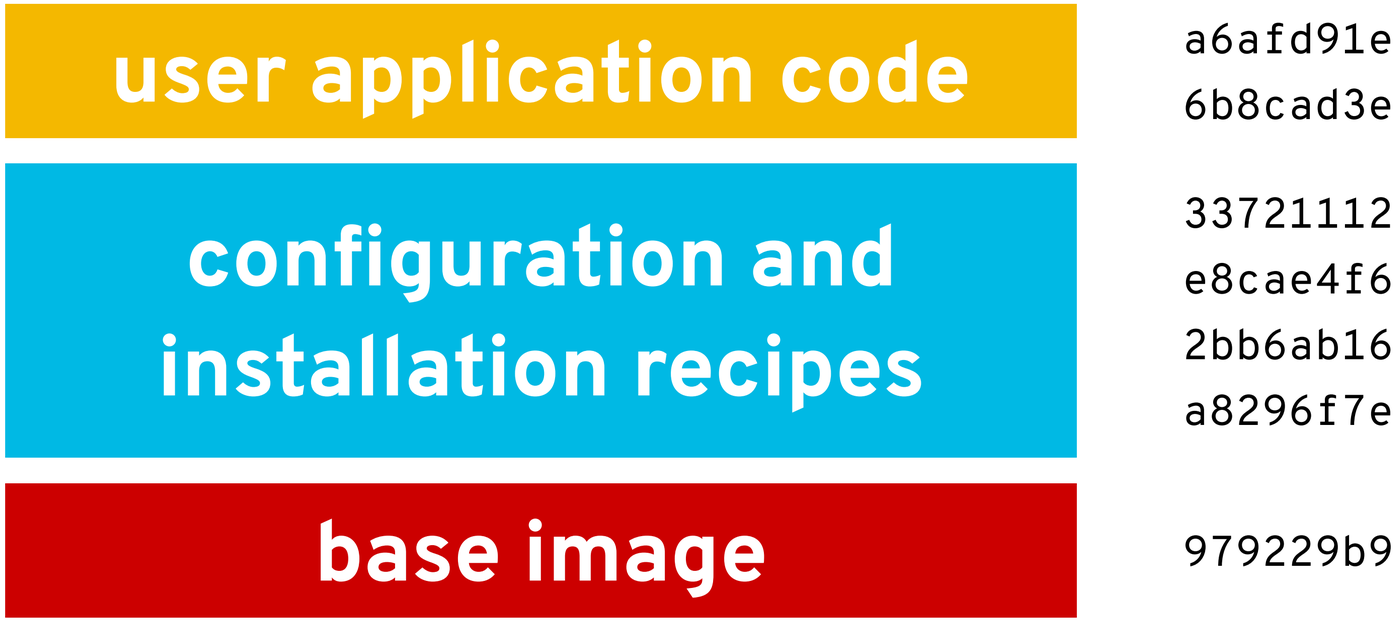
What does Kubernetes build on containers?

Red Hat

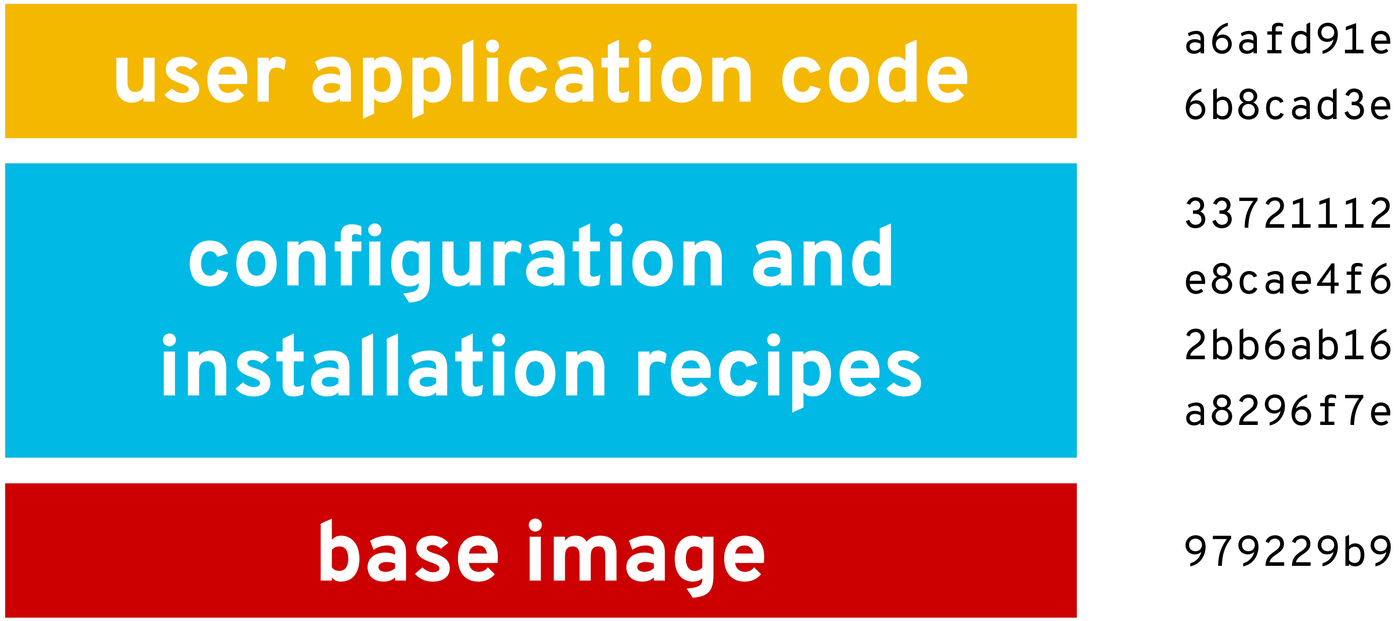
Immutable images

Immutable images

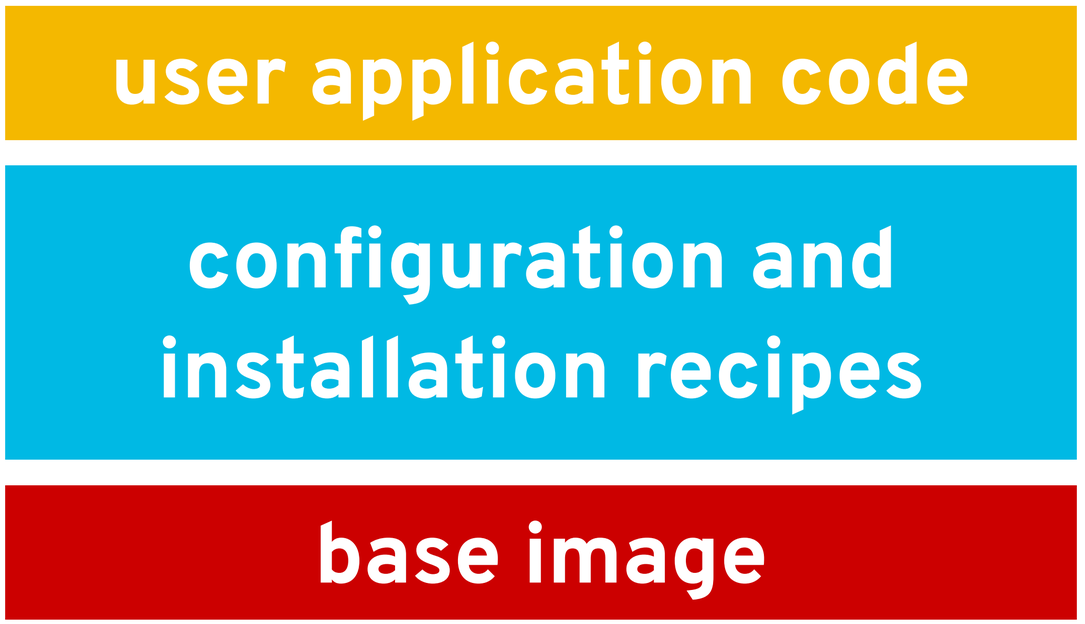
Immutable images



Immutable images



Immutable images



model in production on 16 July 2019

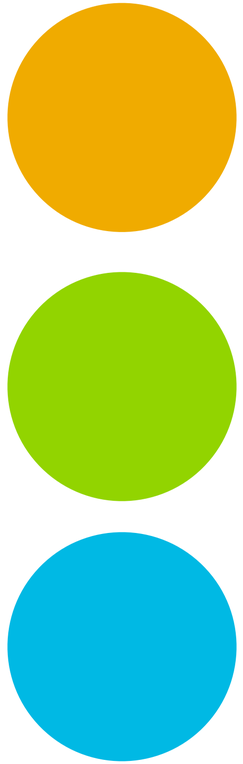
a6afd91e
6b8cad3e

33721112
e8cae4f6
2bb6ab16
a8296f7e

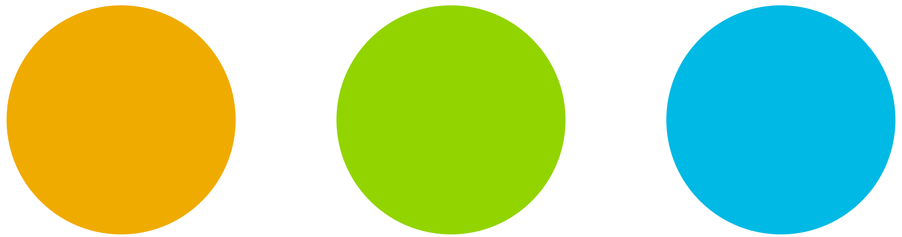
979229b9

Red Hat

Stateless microservices

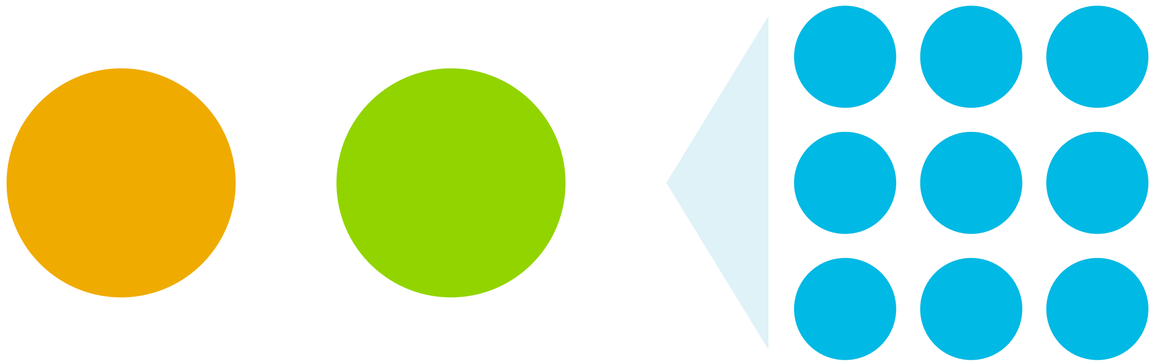


Stateless microservices

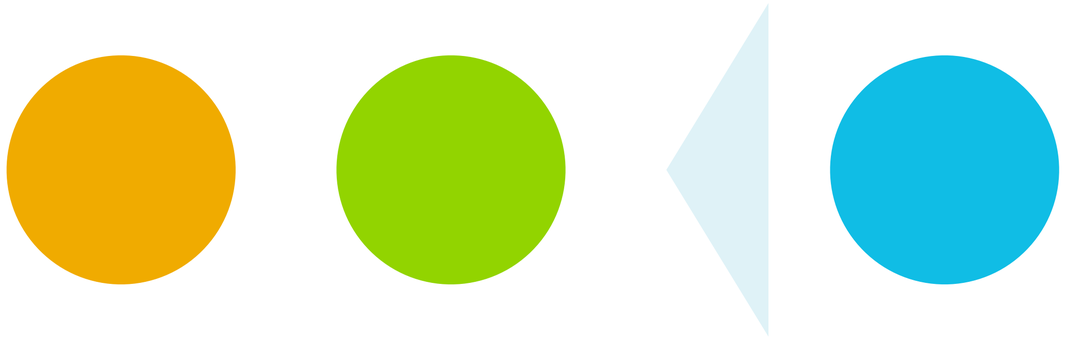


Stateless microservices

Stateless microservices



Stateless microservices

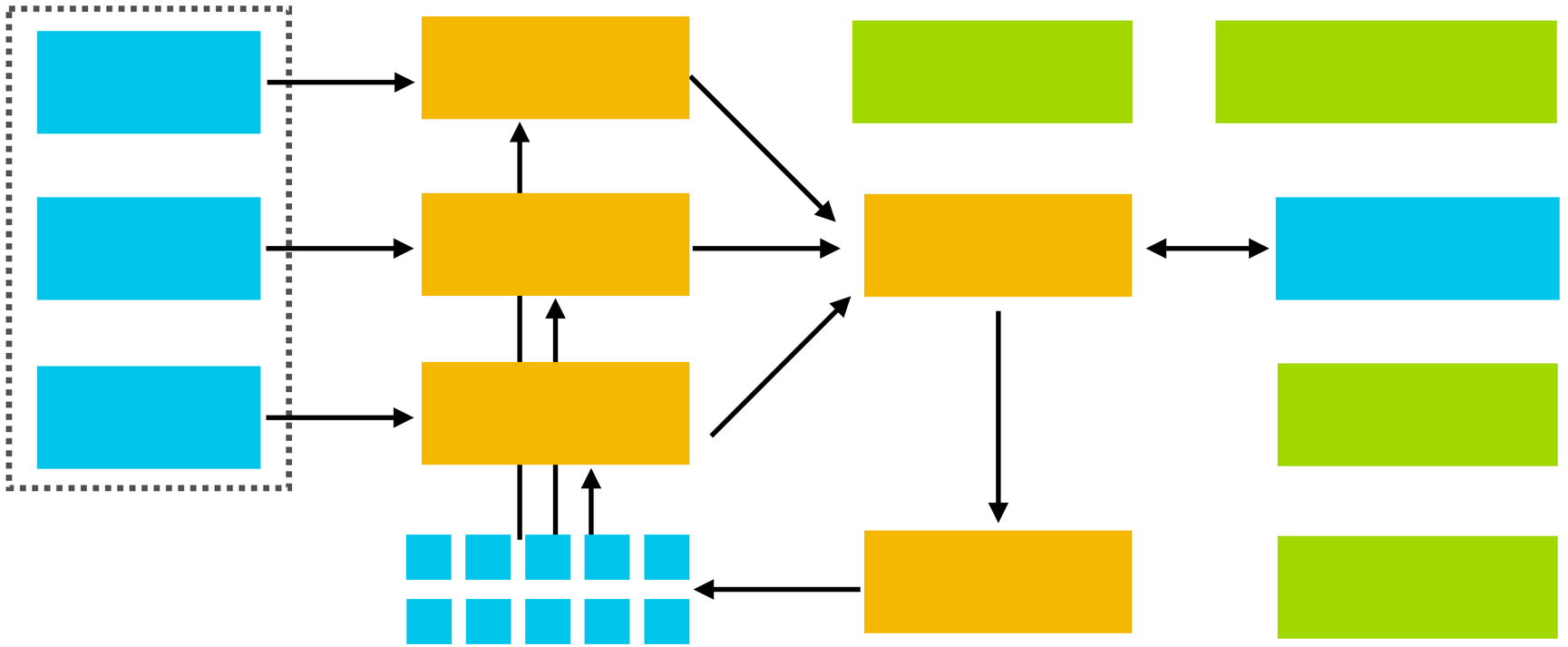


Stateless microservices

Stateless microservices

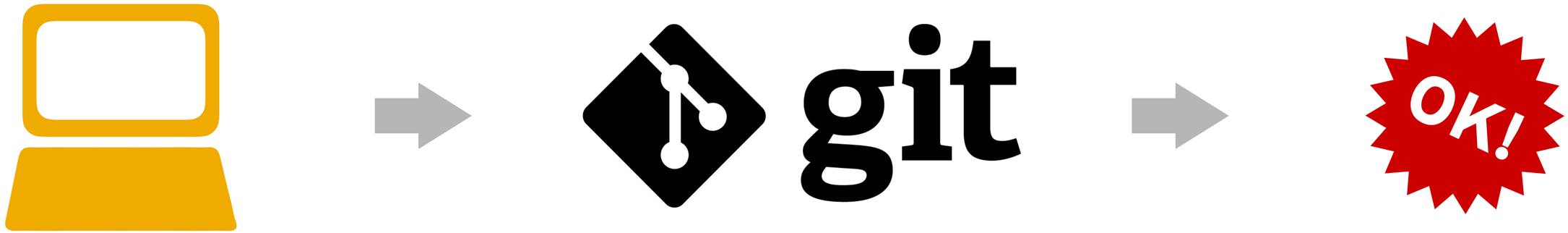
Stateless microservices

Declarative app configuration

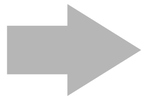


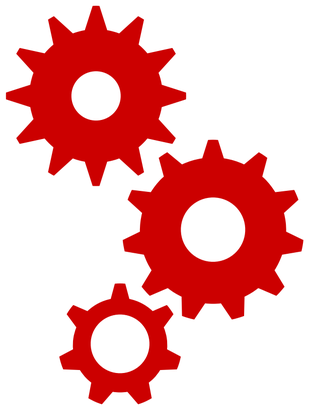
Red Hat

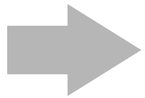
Integration and deployment



Integration and deployment





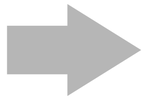


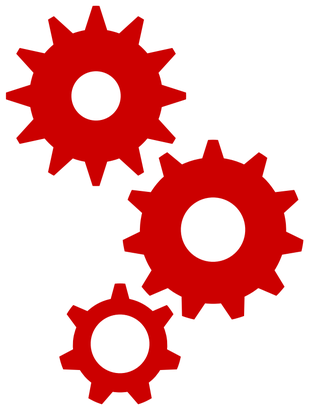
application code

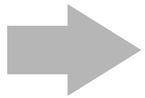
configuration and installation recipes

base image

Integration and deployment







application code

configuration and installation recipes

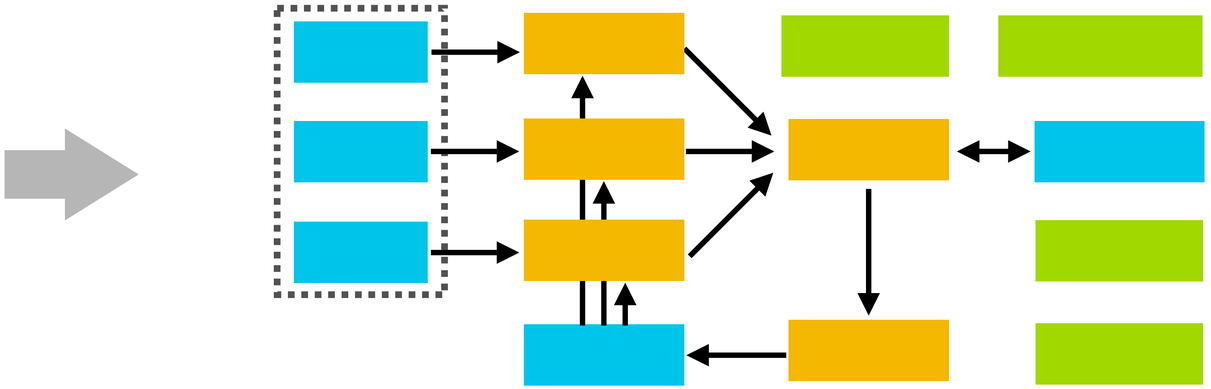
base image

Integration and deployment

application code

configuration and installation recipes

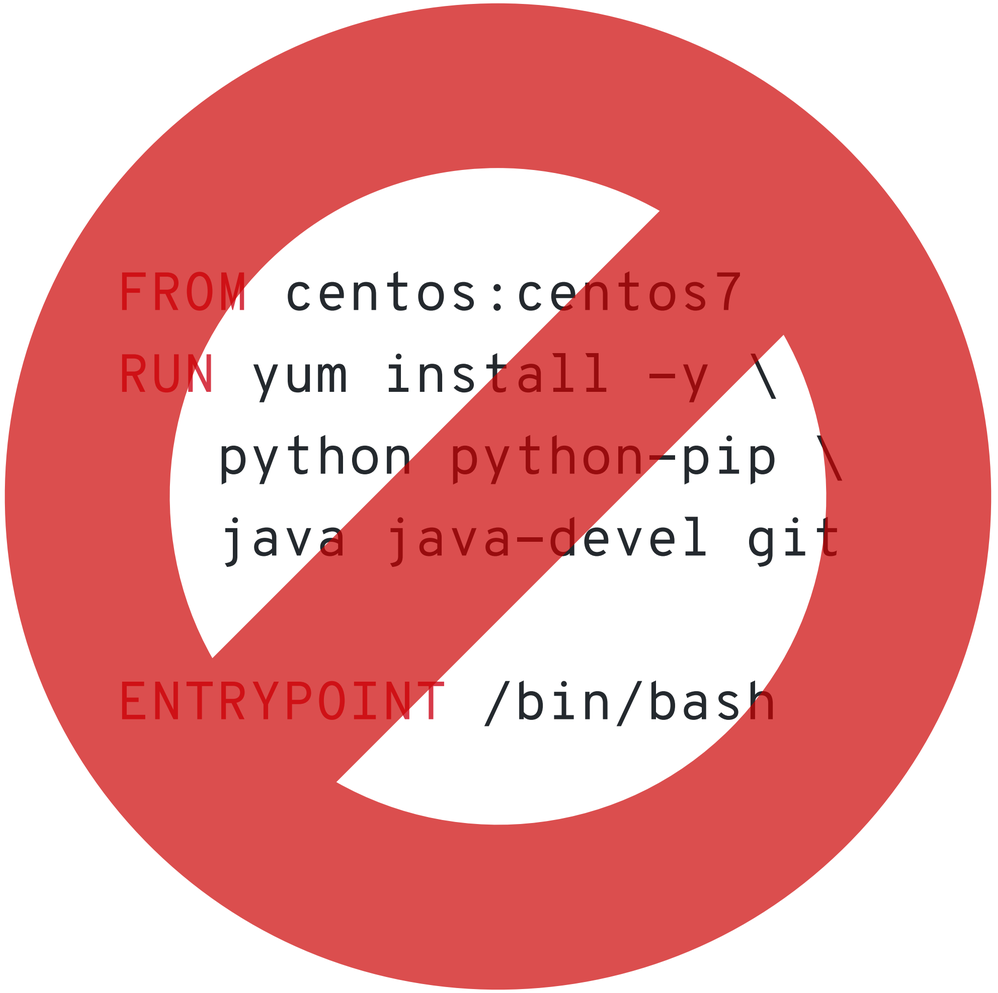
base image

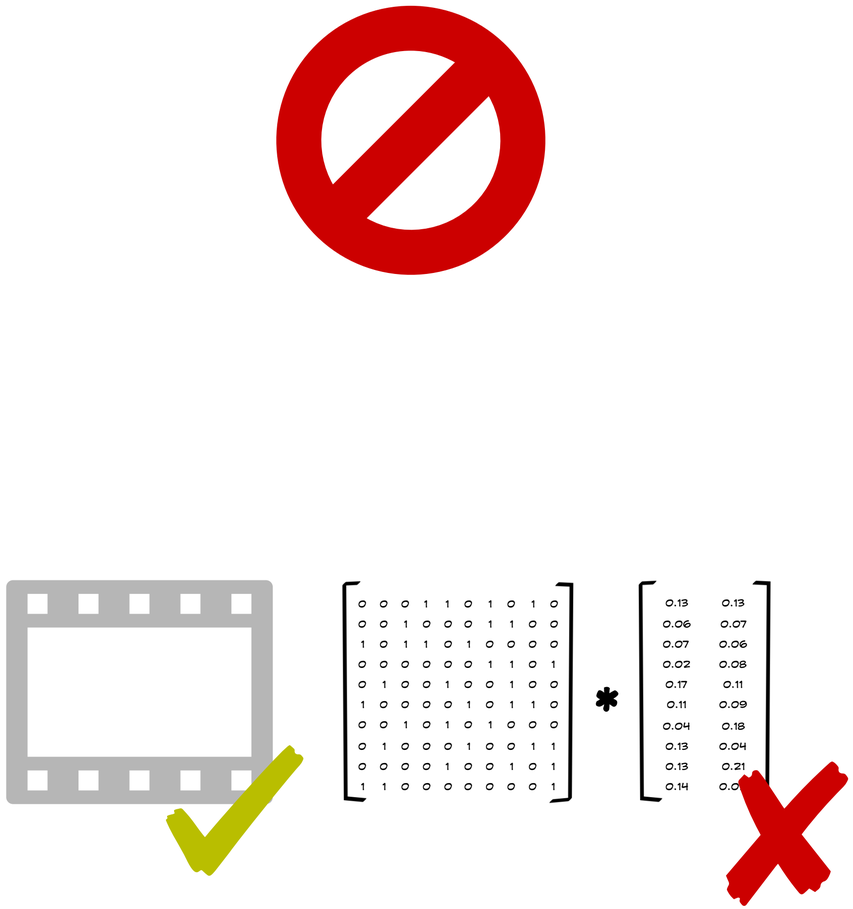


What containers offer for machine learning workflows

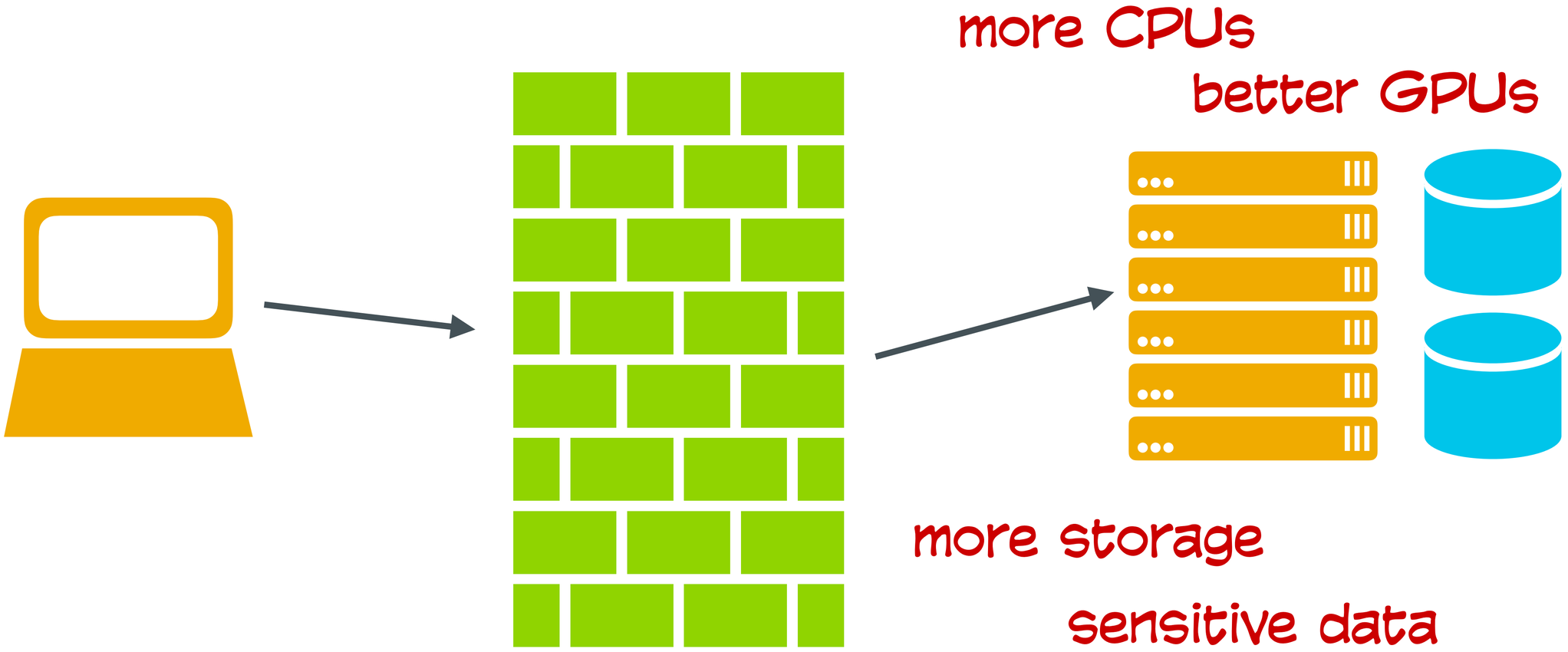
Red Hat

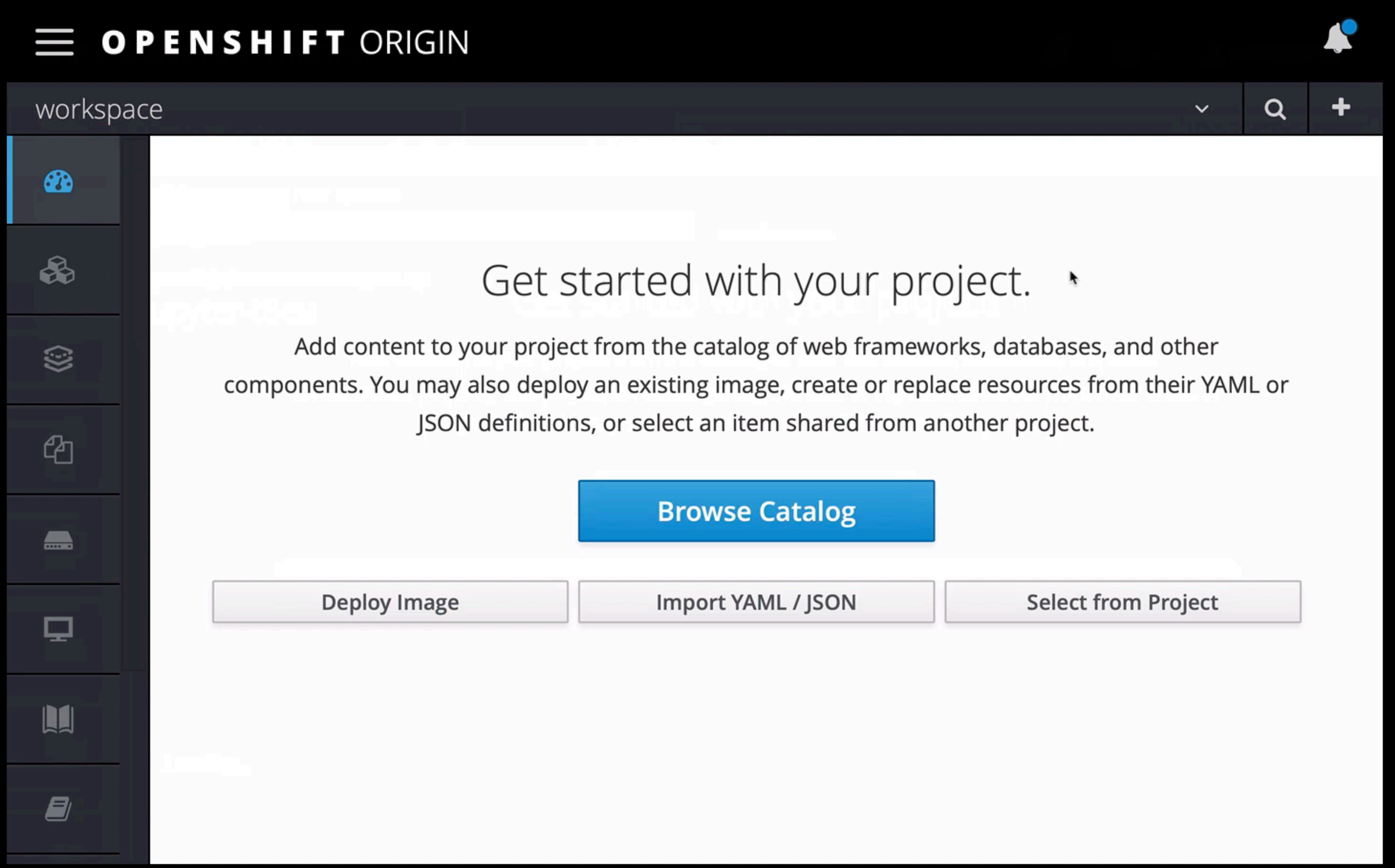
```
FROM centos:centos7
RUN yum install -y \
    python python-pip \
    java java-devel git

ENTRYPOINT /bin/bash
```

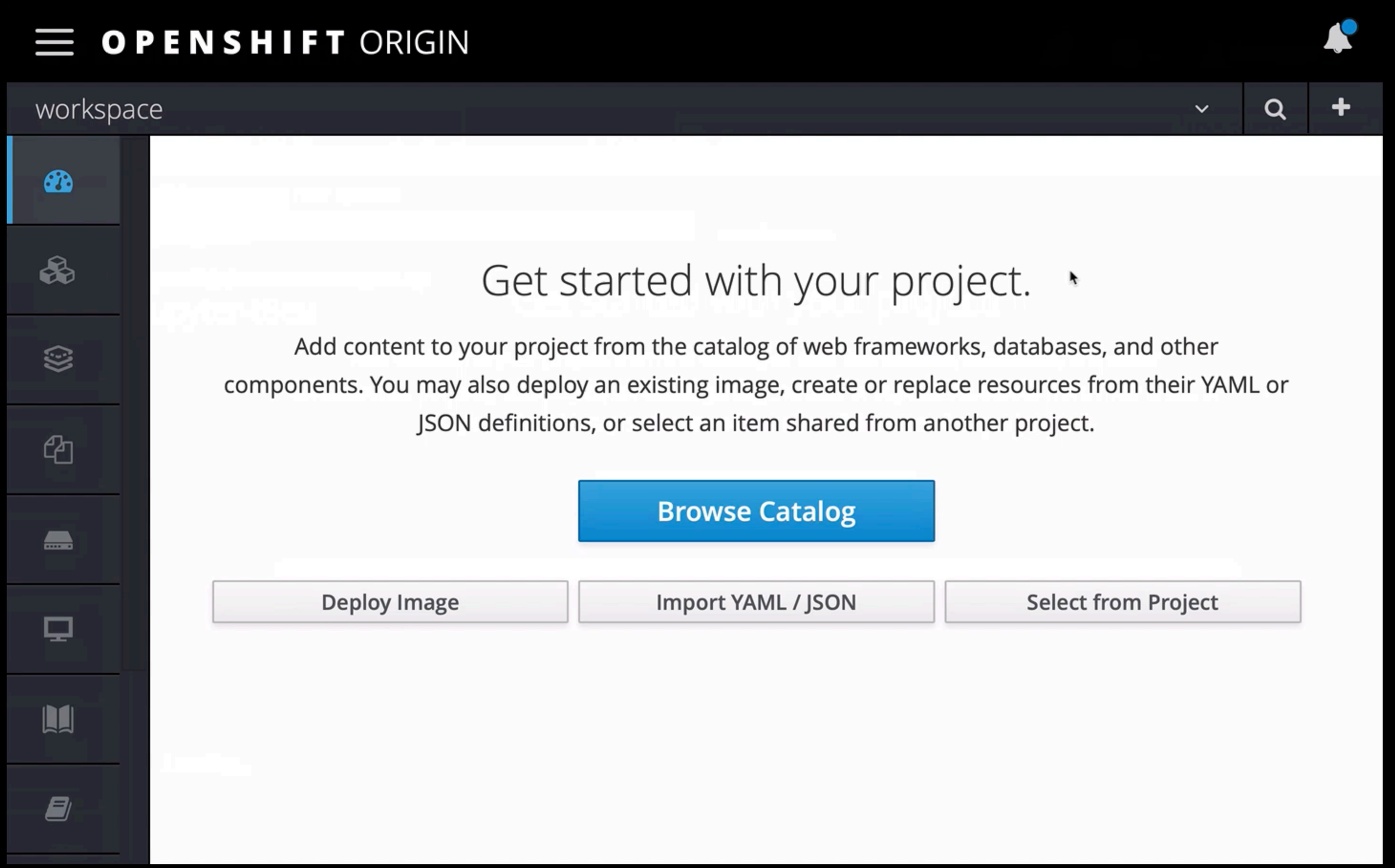

FROM centos:centos7
RUN yum install -y \
python python-pip \
java java-devel git
ENTRYPOINT /bin/bash

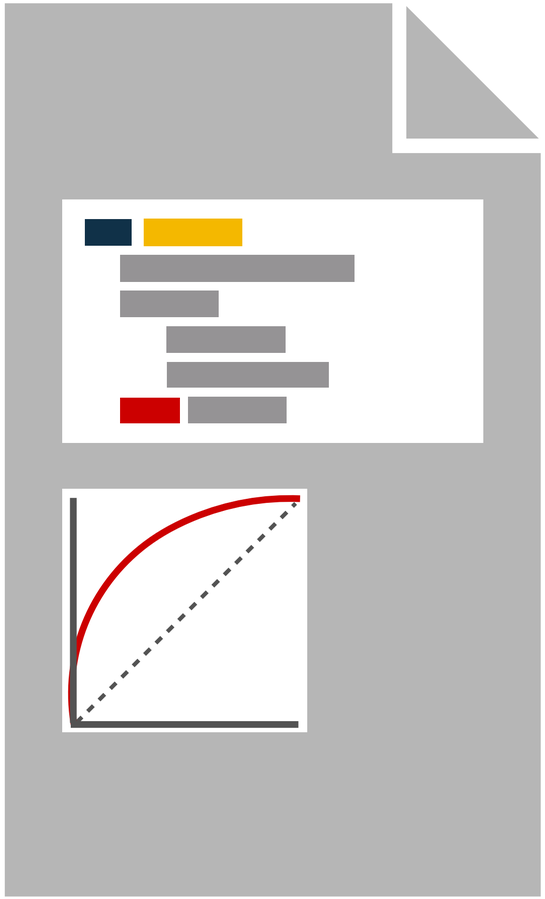

0 0 0 1 1 0 1 0 1 0
0 0 1 0 0 0 1 1 0 0
1 0 1 1 0 1 0 0 0 0
0 0 0 0 0 0 1 1 0 1
0 1 0 0 1 0 0 1 0 0
1 0 0 0 0 1 0 1 1 0
0 0 1 0 1 0 1 0 0 0
0 1 0 0 0 1 0 0 1 1
0 0 0 0 1 0 0 1 0 1
1 1 0 0 0 0 0 0 0 1
*
0.13 0.13
0.06 0.07
0.07 0.06
0.02 0.08
0.17 0.11
0.11 0.09
0.04 0.18
0.13 0.04
0.13 0.21
0.14 0.0

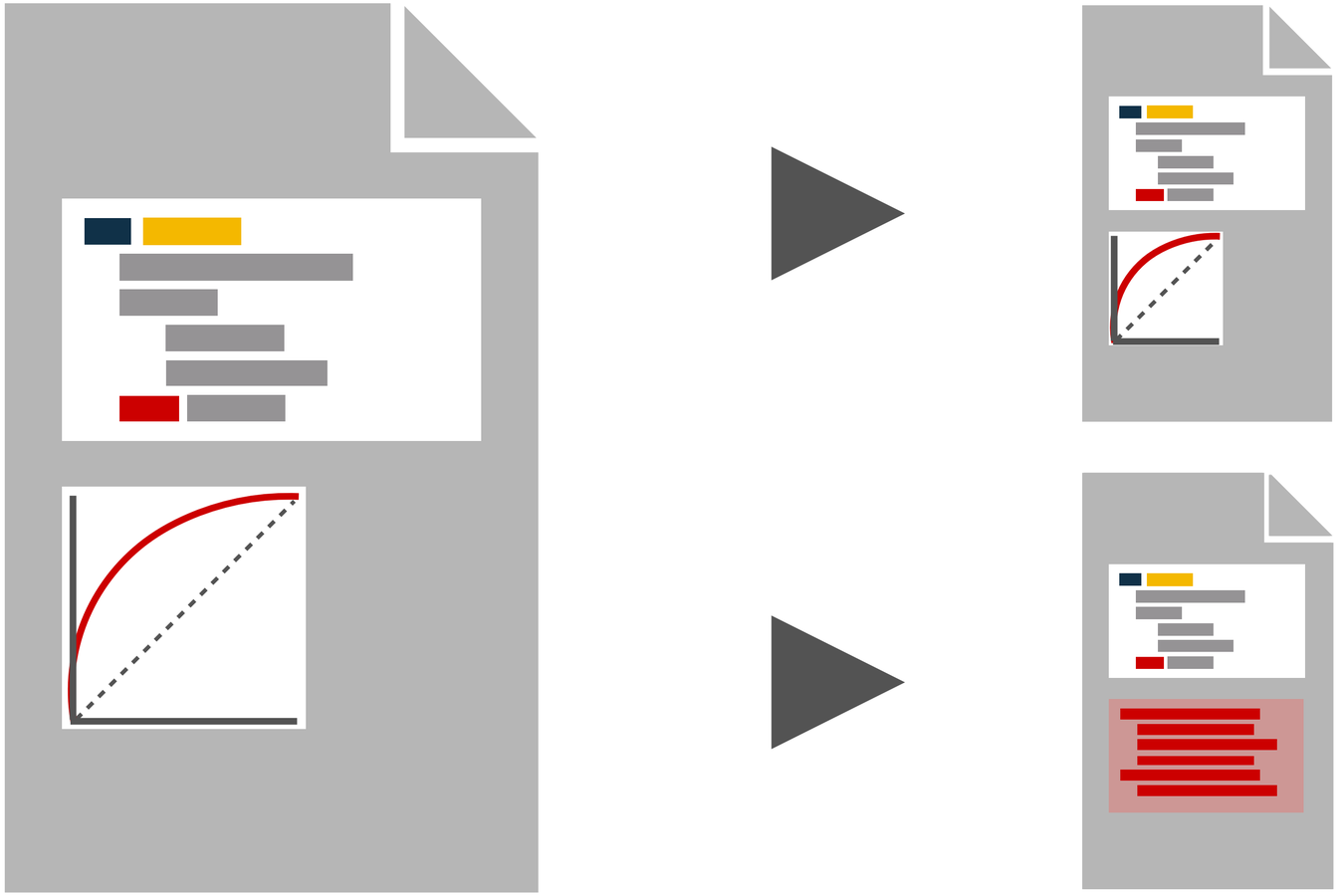
Self-service environments

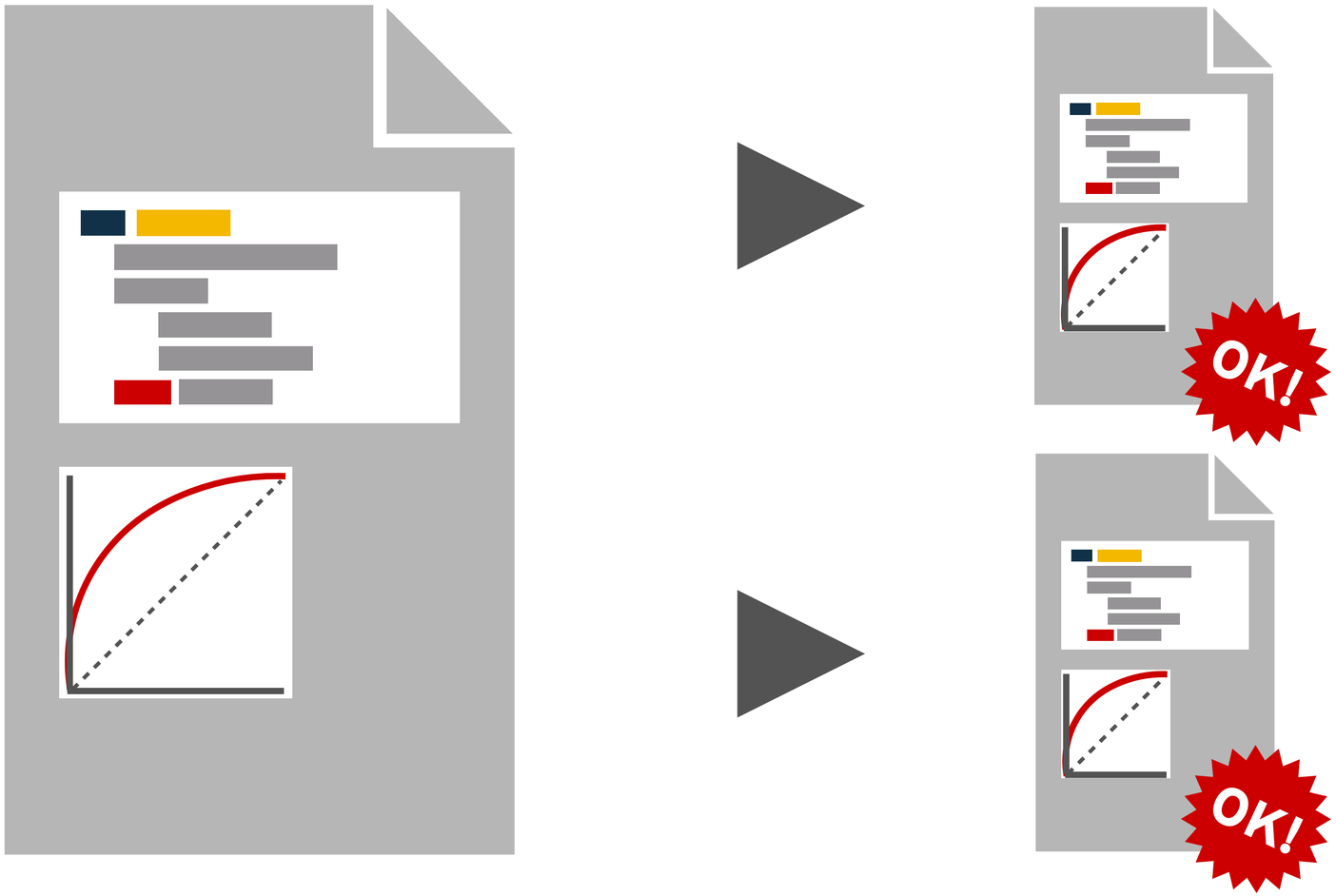



OPENSHIFT ORIGIN
workspace
Get started with your project.
Add content to your project from the catalog of web frameworks, databases, and other components. You may also deploy an existing image, create or replace resources from their YAML or JSON definitions, or select an item shared from another project.
Browse Catalog
Deploy Image
Import YAML / JSON
Select from Project

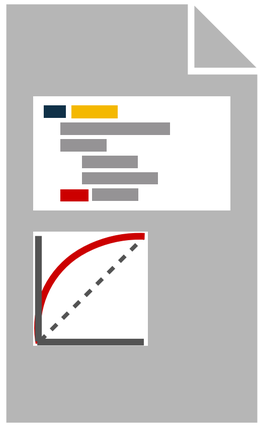

OPENSHIFT ORIGIN
workspace
Get started with your project.
Add content to your project from the catalog of web frameworks, databases, and other components. You may also deploy an existing image, create or replace resources from their YAML or JSON definitions, or select an item shared from another project.
Browse Catalog
Deploy Image
Import YAML / JSON
Select from Project

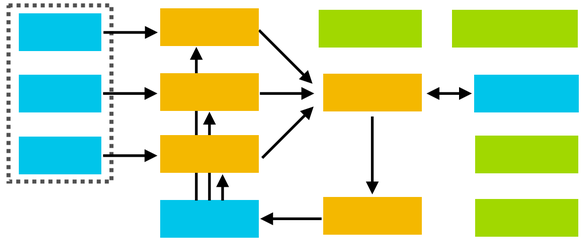




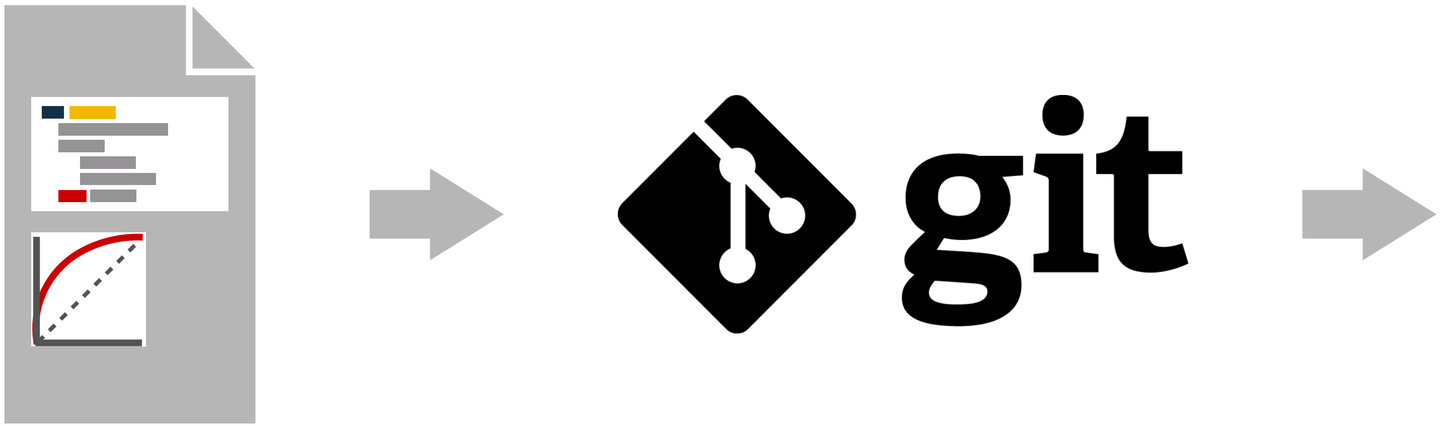

OK!
OK!

No friction: mybinder.org

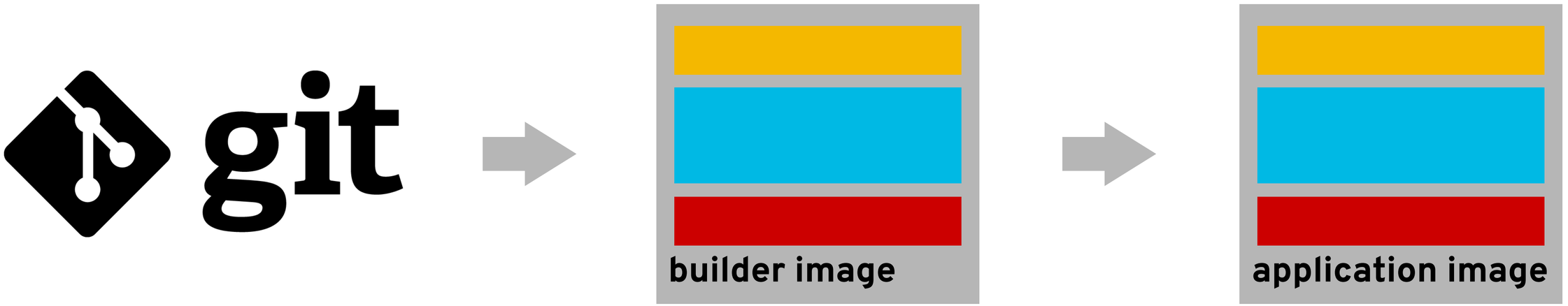




More flexible: source-to-image



More flexible: source-to-image



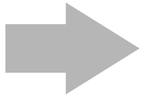
https://github.com/openshift/source-to-image

```
oc new-app getwarped/s2i-minimal-notebook:latest~\
   https://github.com/willb/probabilistic-structures \
   -e JUPYTER_NOTEBOOK_PASSWORD=developer
```

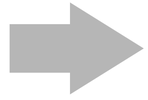
```
oc new-app getwarped/s2i-minimal-notebook:latest~\
   https://github.com/willb/probabilistic-structures \
   -e JUPYTER_NOTEBOOK_PASSWORD=developer
```

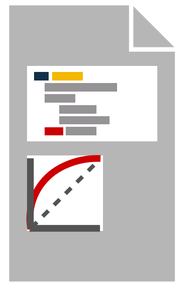
```
oc new-app getwarped/s2i-minimal-notebook:latest~\
   https://github.com/willb/probabilistic-structures \
   -e JUPYTER_NOTEBOOK_PASSWORD=developer
```

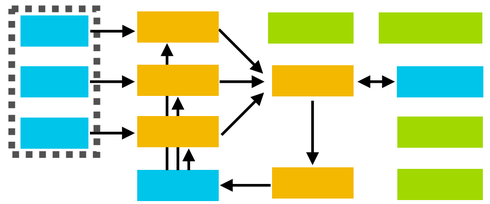
```
oc new-app getwarped/s2i-minimal-notebook:latest~\
    https://github.com/willb/probabilistic-structures \
    -e JUPYTER_NOTEBOOK_PASSWORD=developer
```

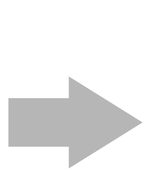


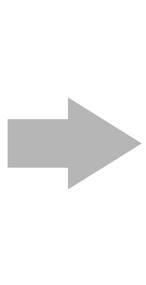

git

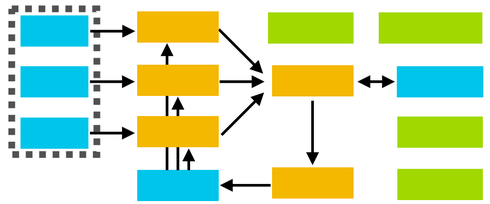


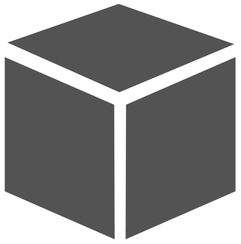


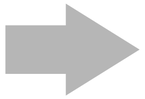


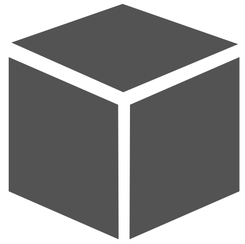


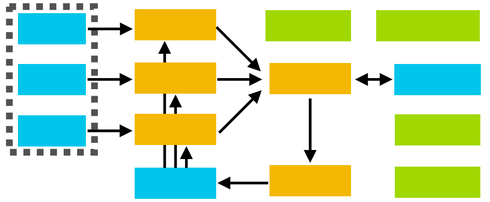




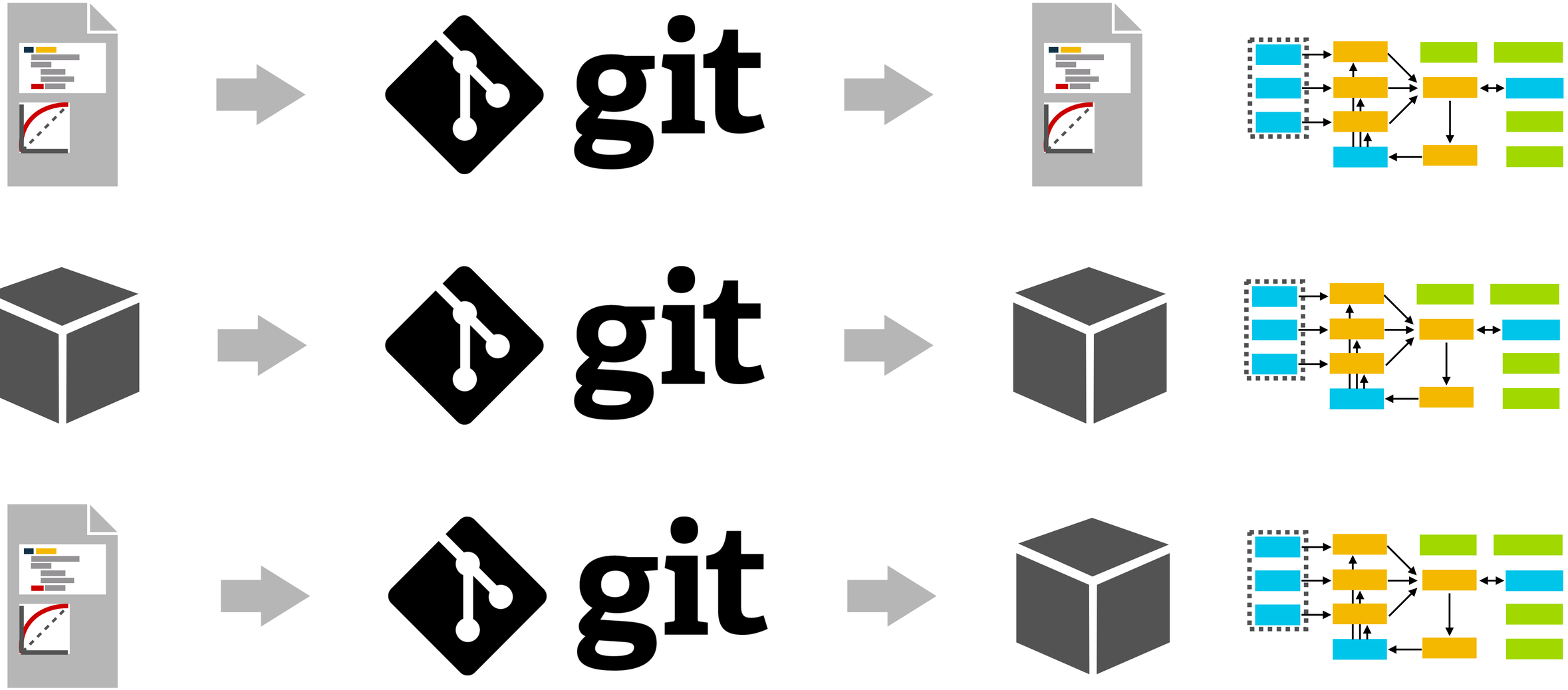








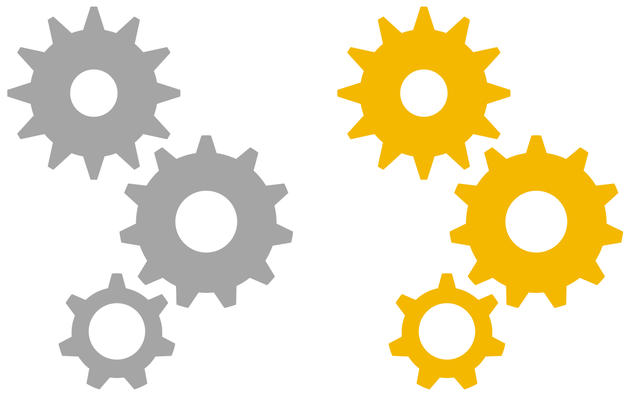
Red Hat

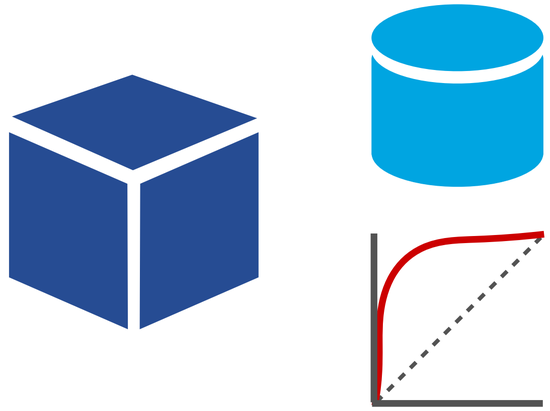

git
git
git

```
willb@echo % oc new-app --name model \
quay.io/willbenton/simple-model-s2i:demo\
~https://github.com/willb/example-model-s2i-notebook
```

```
willb@echo % oc new-app --name model \
quay.io/willbenton/simple-model-s2i:demo\
~https://github.com/willb/example-model-s2i-notebook
```

codifying problem and metrics ▶ data collection and cleaning ▶ **feature engineering** ▶ **model training and tuning** ▶ model validation ▶ **model deployment** ▶ **monitoring, validation**





codifying problem and metrics

data collection and cleaning

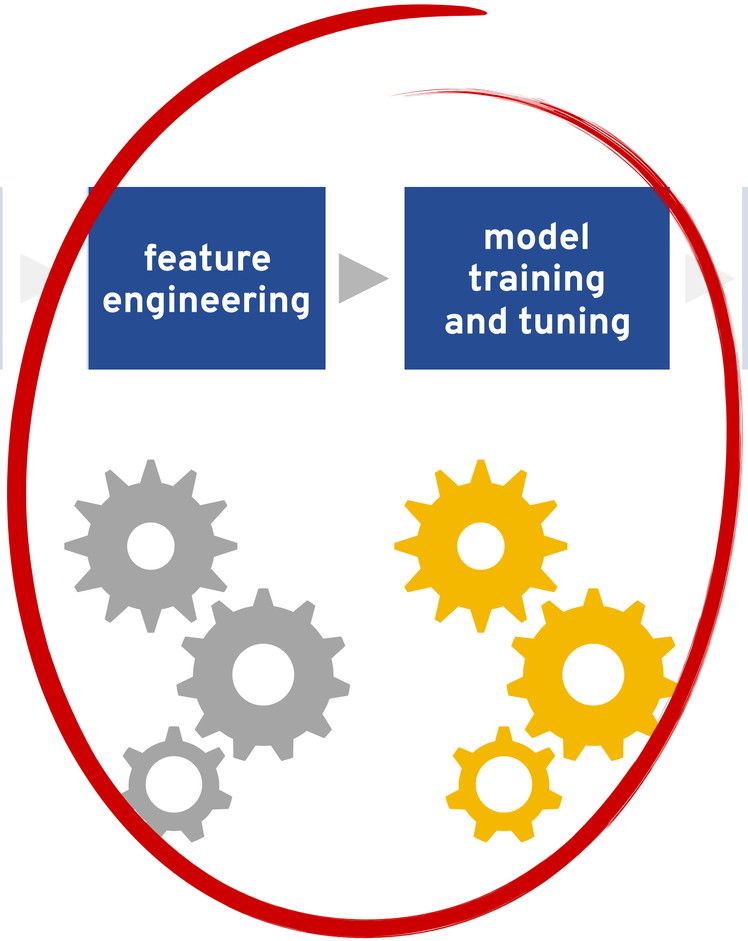
feature engineering

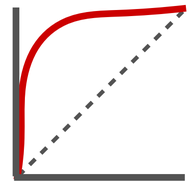
model training and tuning

model validation

model deployment

monitoring, validation





In this notebook we will process the synthetic Austen/food reviews data and convert it into feature vectors. In later notebooks these feature vectors will be the inputs to models which we will train and eventually use to identify spam.

This notebook uses term frequency-inverse document frequency, or tf-idf, to generate feature vectors. Tf-idf is commonly used to summarise text data, and it aims to capture how important different words are within a set of documents. Tf-idf combines a normalized word count (or term frequency) with the inverse document frequency (or a measure of how common a word is across all documents) in order to identify words, or terms, which are 'interesting' or important within the document.

We begin by loading in the data:

In [1]:
```
import pandas as pd

df = pd.read_parquet("data/training.parquet")
```

To illustrate the computation of tf-idf vectors we will first implement the method on a sample of

In this notebook we will process the synthetic Austen/food reviews data and convert it into feature vectors. In later notebooks these feature vectors will be the inputs to models which we will train and eventually use to identify spam.

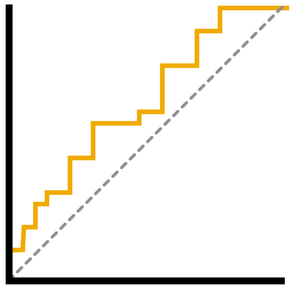
This notebook uses term frequency-inverse document frequency, or tf-idf, to generate feature vectors. Tf-idf is commonly used to summarise text data, and it aims to capture how important different words are within a set of documents. Tf-idf combines a normalized word count (or term frequency) with the inverse document frequency (or a measure of how common a word is across all documents) in order to identify words, or terms, which are 'interesting' or important within the document.

We begin by loading in the data:

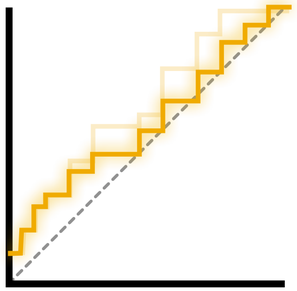
In [1]:
```
import pandas as pd

df = pd.read_parquet("data/training.parquet")
```

To illustrate the computation of tf-idf vectors we will first implement the method on a sample of


mA

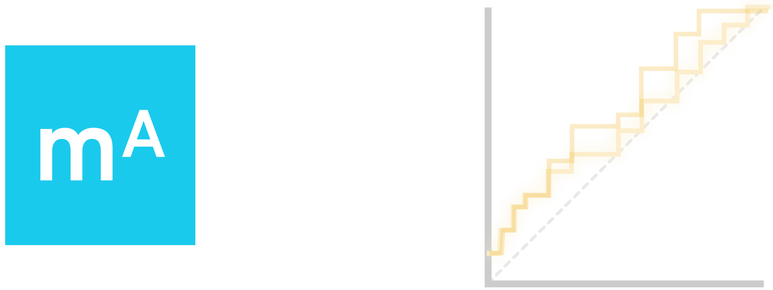



mA



(joint) distribution of input data?

distribution of predictions?



distribution of number of multiplications while scoring?

OpenShift Web Console | 07-services | Prometheus Time Series Colle

Not Secure | jupyter-d1pa-myproject.127.0.0.1.nip.io/notebooks/07-services.ipynb

jupyter 07-services Last Checkpoint: a minute ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

200 rows × 4 columns

Running our models as services gives us an interesting opportunity to detect data drift by publishing the distribution of our predictions as metrics. If the distribution of predictions shifts over time, we can use that as an indication that the distribution of the data we're evaluating has shifted as well, and that we should re-train our model.
In this example, our pipeline service publishes metrics related to the predictions made by the model (keys beginning with `pipeline_predictions_`) as well as metrics related to the computational performance of our pipeline service (keys beginning with `pipeline_processing_seconds_`).

In [ ]:
```
get_metrics()
```

Since our service publishes Prometheus metrics, we can define alerting rules or visualize how our metric values change over time.

In [ ]:
```
def experiment(data, size, **kwargs):
    for k, v in kwargs.items():
        sample = data[data.label == k].sample(int(size * v))
        score_text(sample["text"].values.tolist())
```

In [ ]:
```
experiment(data, 20000, legitimate=.05, spam=.95)
experiment(data, 20000, legitimate=.05, spam=.95)
experiment(data, 20000, legitimate=.05, spam=.95)
```

In [ ]:
```
experiment(data, 20000, legitimate=.25, spam=.75)
```

In [ ]:
```
experiment(data, 20000, legitimate=1)
```

Exercises

1. What would a REST API for serving multiple versions of multiple pipelines look like? (Hint: consider the verbs and nouns involved.)
2. While we don't have a monitoring stack enabled for this workshop, you can certainly install and configure one in your own OpenShift cluster. What sort of alerting rules might you install to identify data drift in this application?

OpenShift Web Console | 07-services | Prometheus Time Series Colle

Not Secure | jupyter-d1pa-myproject.127.0.0.1.nip.io/notebooks/07-services.ipynb

jupyter 07-services Last Checkpoint: a minute ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

200 rows × 4 columns

Running our models as services gives us an interesting opportunity to detect data drift by publishing the distribution of our predictions as metrics. If the distribution of predictions shifts over time, we can use that as an indication that the distribution of the data we're evaluating has shifted as well, and that we should re-train our model.
In this example, our pipeline service publishes metrics related to the predictions made by the model (keys beginning with `pipeline_predictions_`) as well as metrics related to the computational performance of our pipeline service (keys beginning with `pipeline_processing_seconds_`).

In [ ]:
```
get_metrics()
```

Since our service publishes Prometheus metrics, we can define alerting rules or visualize how our metric values change over time.

In [ ]:
```
def experiment(data, size, **kwargs):
    for k, v in kwargs.items():
        sample = data[data.label == k].sample(int(size * v))
        score_text(sample["text"].values.tolist())
```

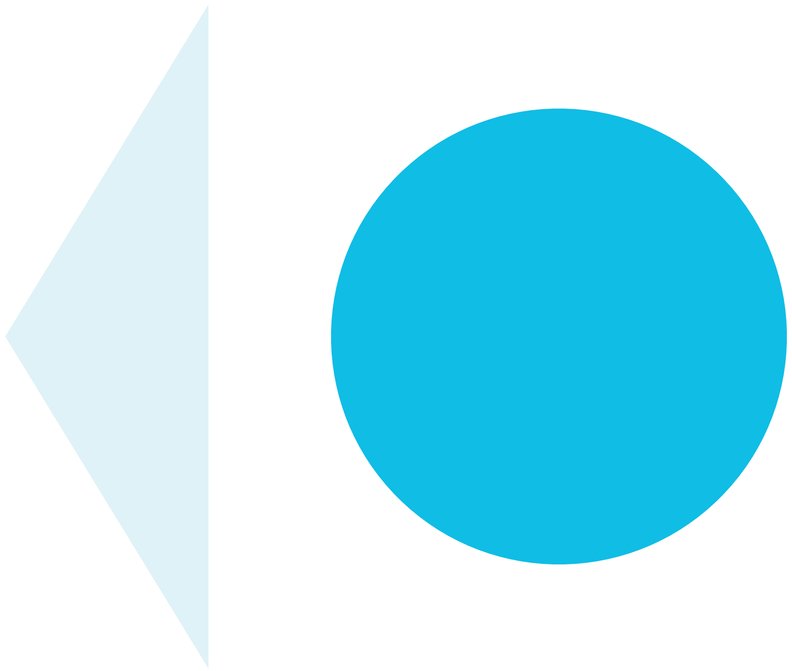
In [ ]:
```
experiment(data, 20000, legitimate=.05, spam=.95)
experiment(data, 20000, legitimate=.05, spam=.95)
experiment(data, 20000, legitimate=.05, spam=.95)
```

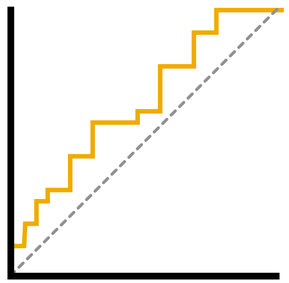
In [ ]:
```
experiment(data, 20000, legitimate=.25, spam=.75)
```

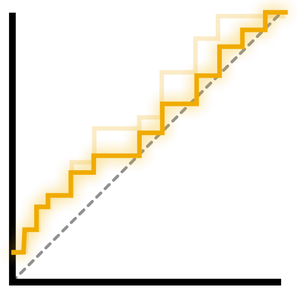
In [ ]:
```
experiment(data, 20000, legitimate=1)
```

Exercises

1. What would a REST API for serving multiple versions of multiple pipelines look like? (Hint: consider the verbs and nouns involved.)
2. While we don't have a monitoring stack enabled for this workshop, you can certainly install and configure one in your own OpenShift cluster. What sort of alerting rules might you install to identify data drift in this application?

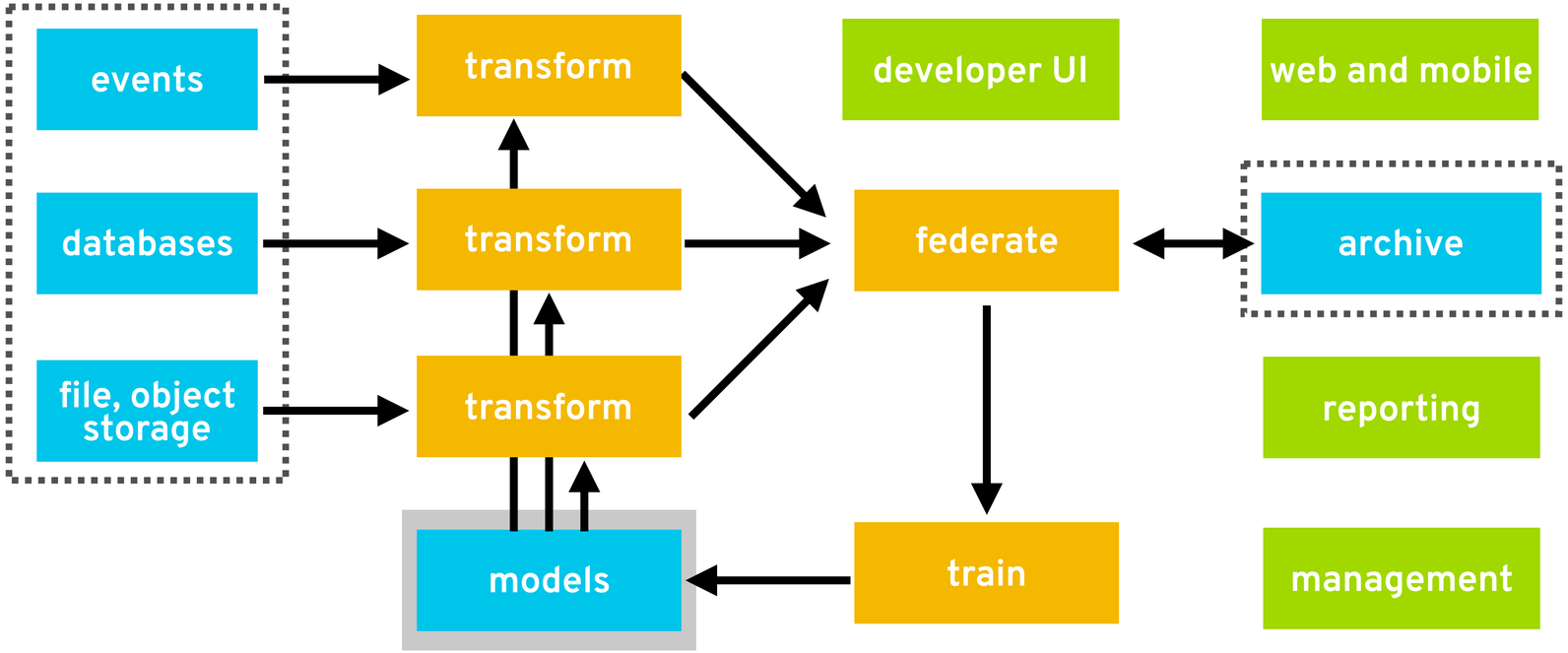


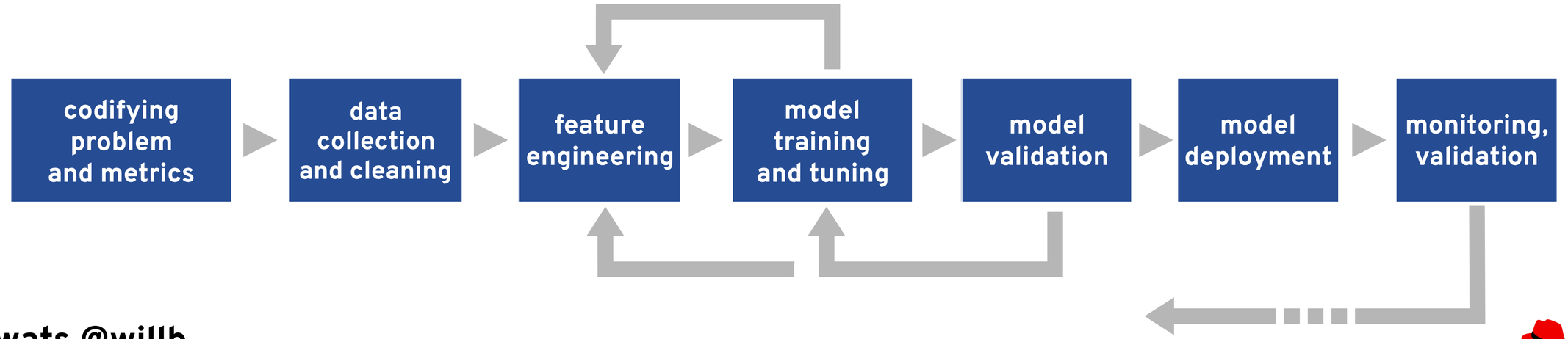




Where from here?

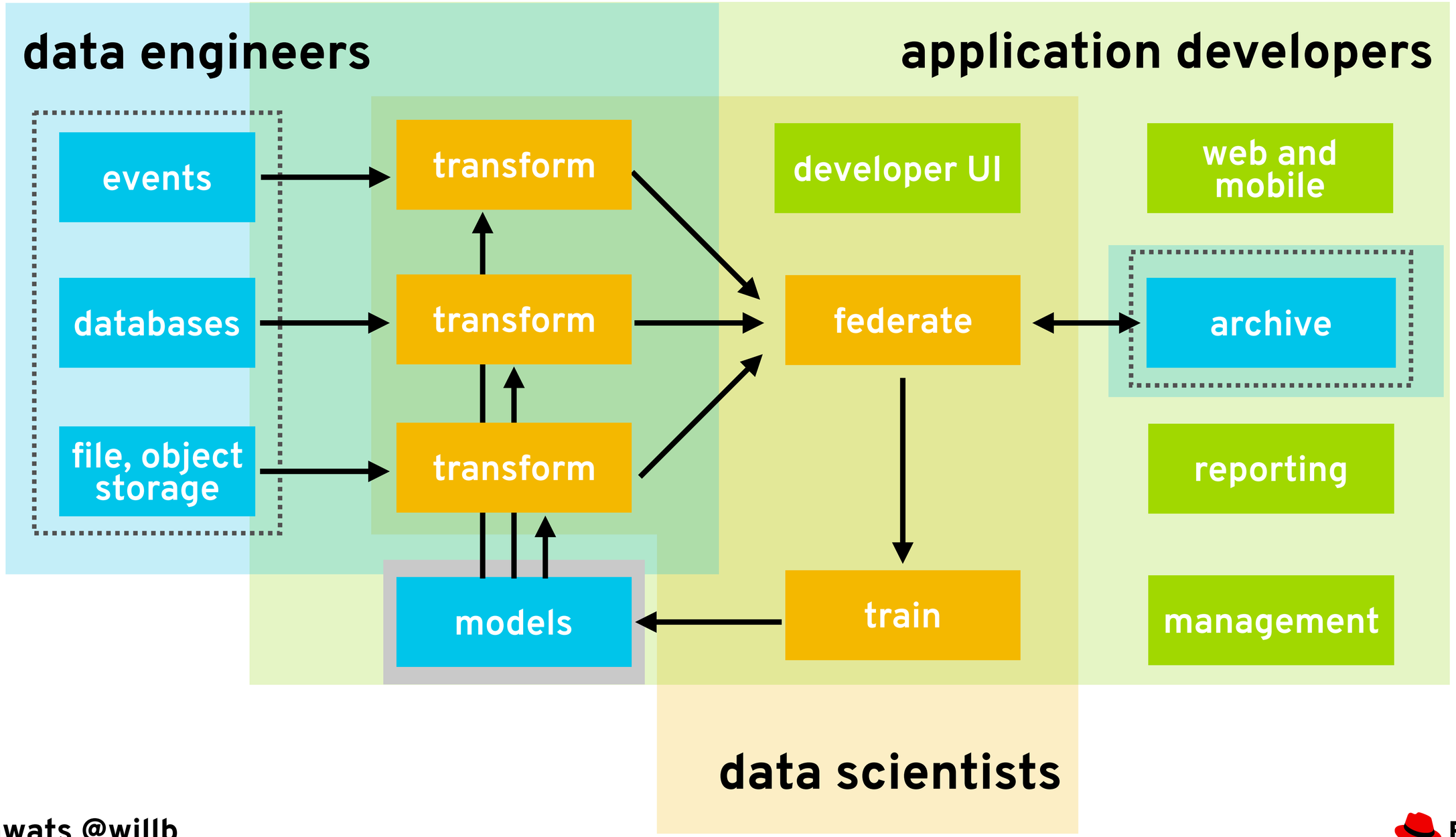
Red Hat

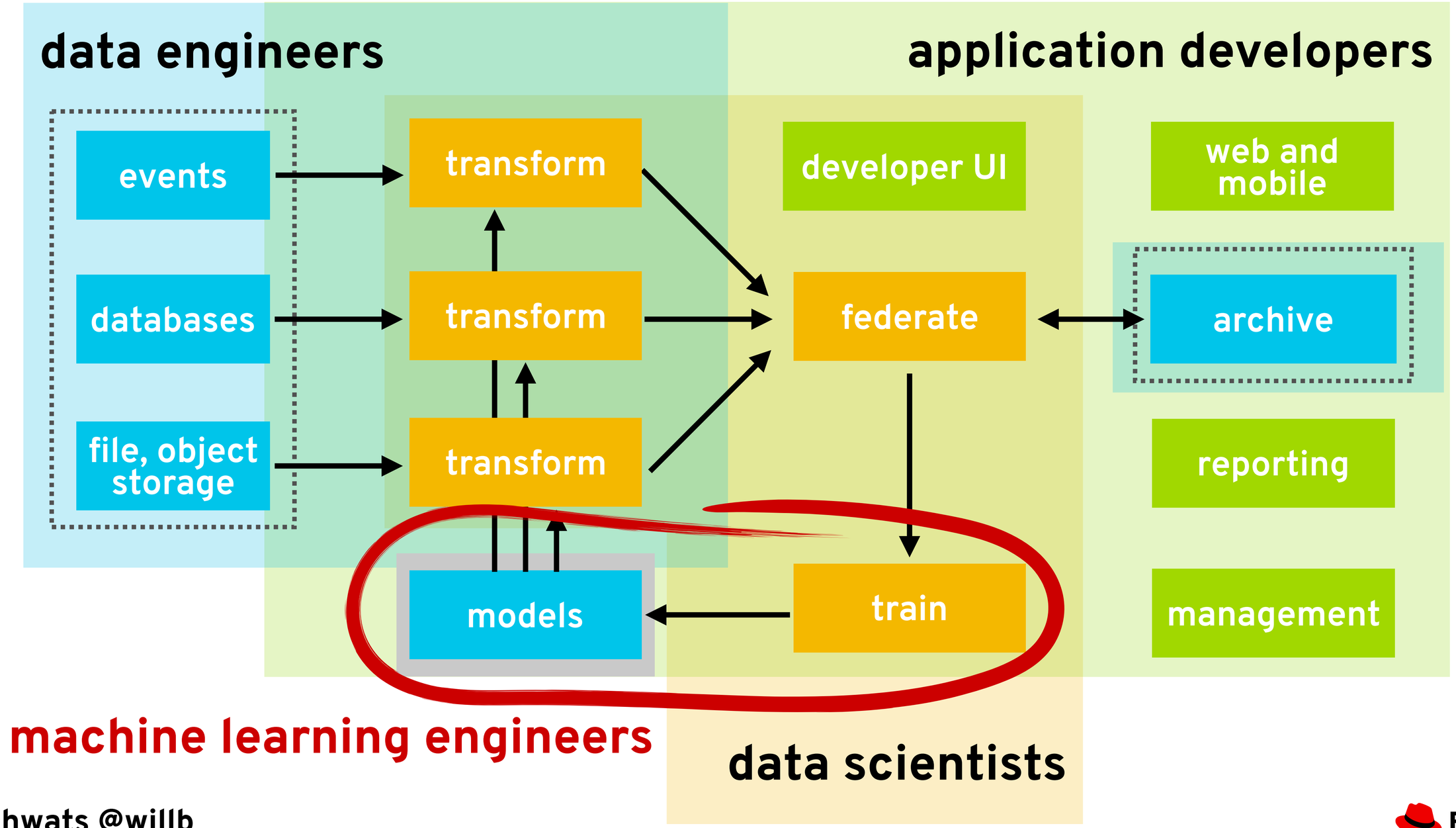




@sophwats @willb

Red Hat

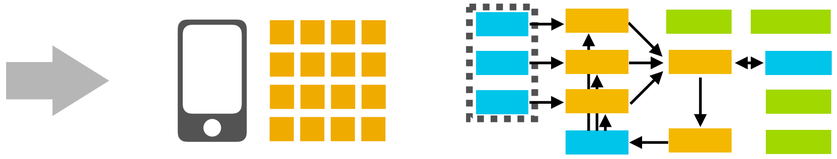

data engineers
application developers
events
databases
file, object storage
transform
transform
transform
models
developer UI
federate
train
web and mobile
archive
reporting
management
data scientists


data engineers
application developers
events
databases
file, object storage
transform
transform
transform
developer UI
federate
web and mobile
archive
reporting
management
models
train
machine learning engineers
data scientists

Red Hat

opendatahub.io

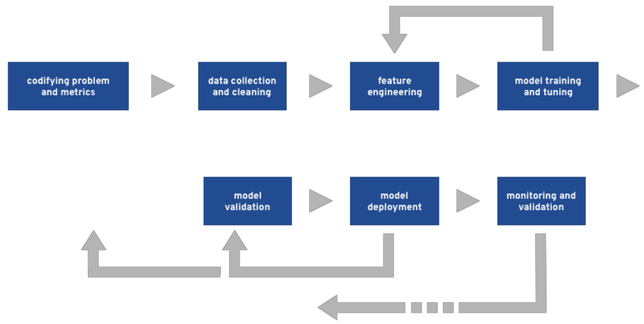
radanalytics.io

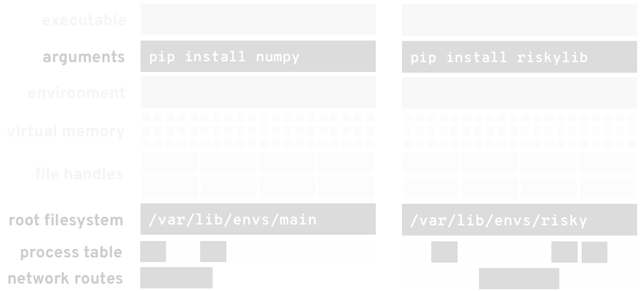


Kubeflow

What did we talk about today?

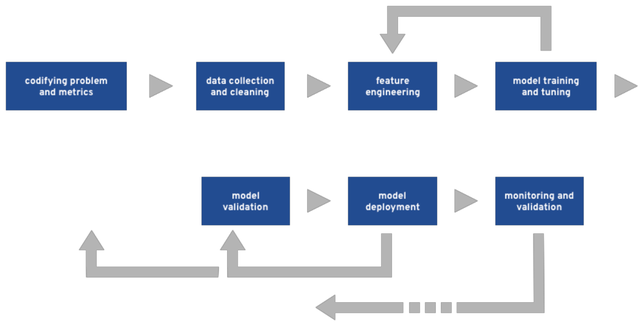
Red Hat

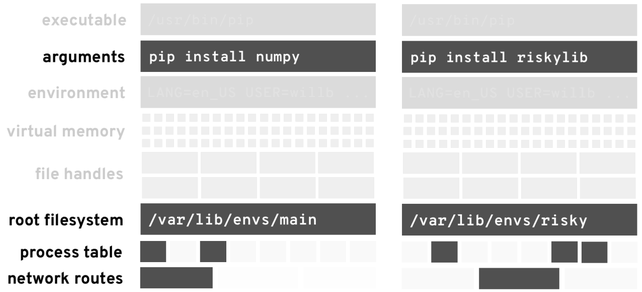




@sophwats @willb

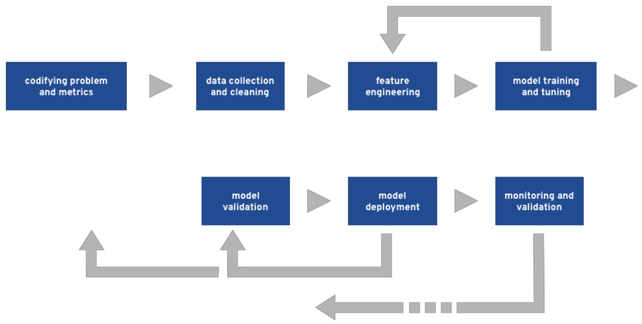
Red Hat

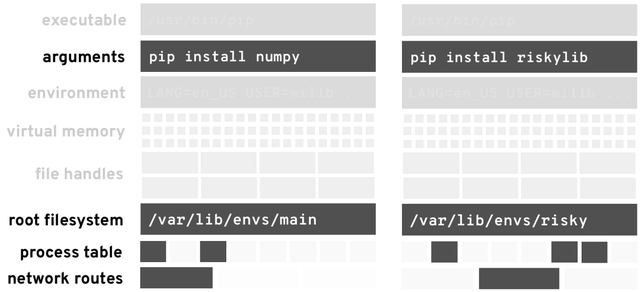




@sophwats @willb

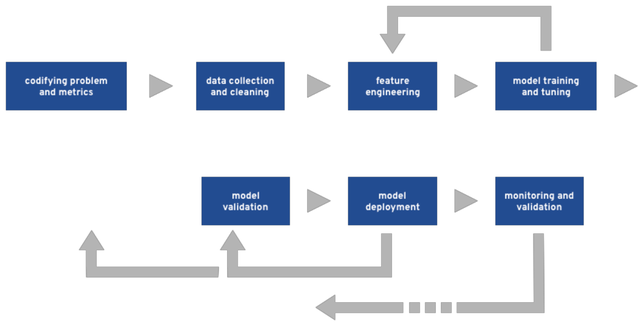
Red Hat

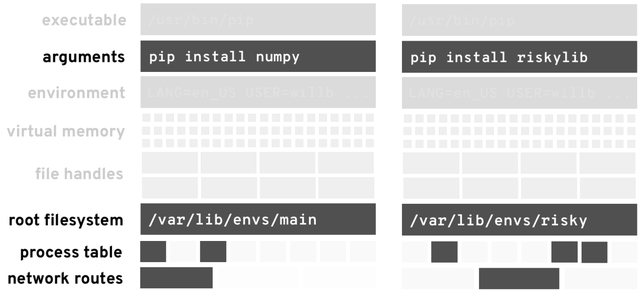


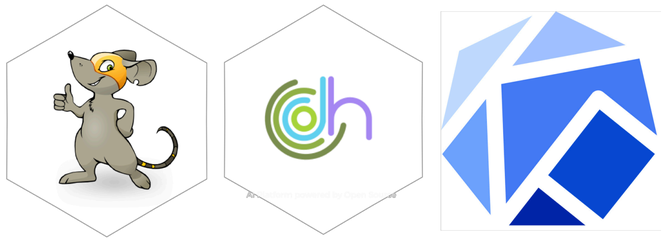


@sophwats @willb

Red Hat

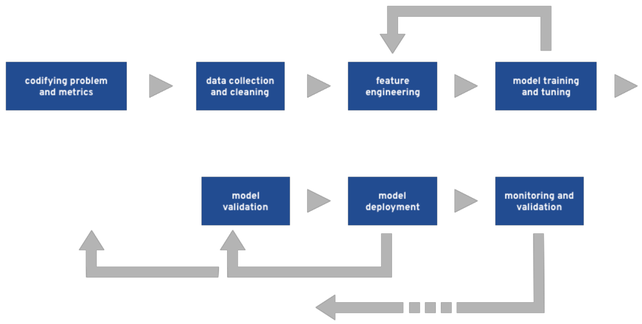


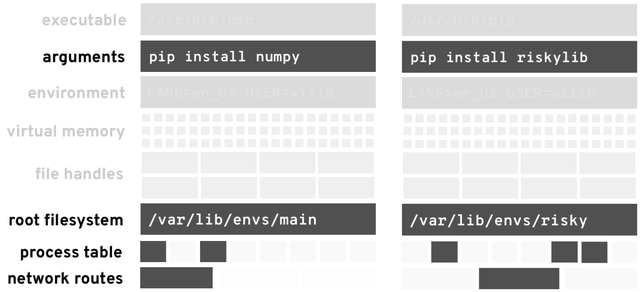


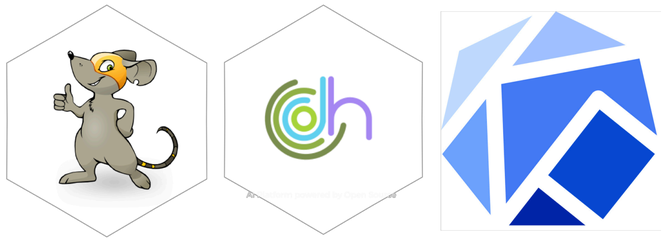


@sophwats @willb

Red Hat







THANKS

sophie@redhat.com • @sophwats
willb@redhat.com • @willb

Red Hat