Seldon Deploy

smv

Oct 18, 2021

CONTENTS

CHAPTER

ONE

ABOUT

1.1 What is Seldon-Deploy?

Seldon Deploy provides oversight and governance for machine learning deployments.

Easily deploy your models in an audited way with gitops. Leverage advanced monitoring and perform alibi-powered explanations on requests.

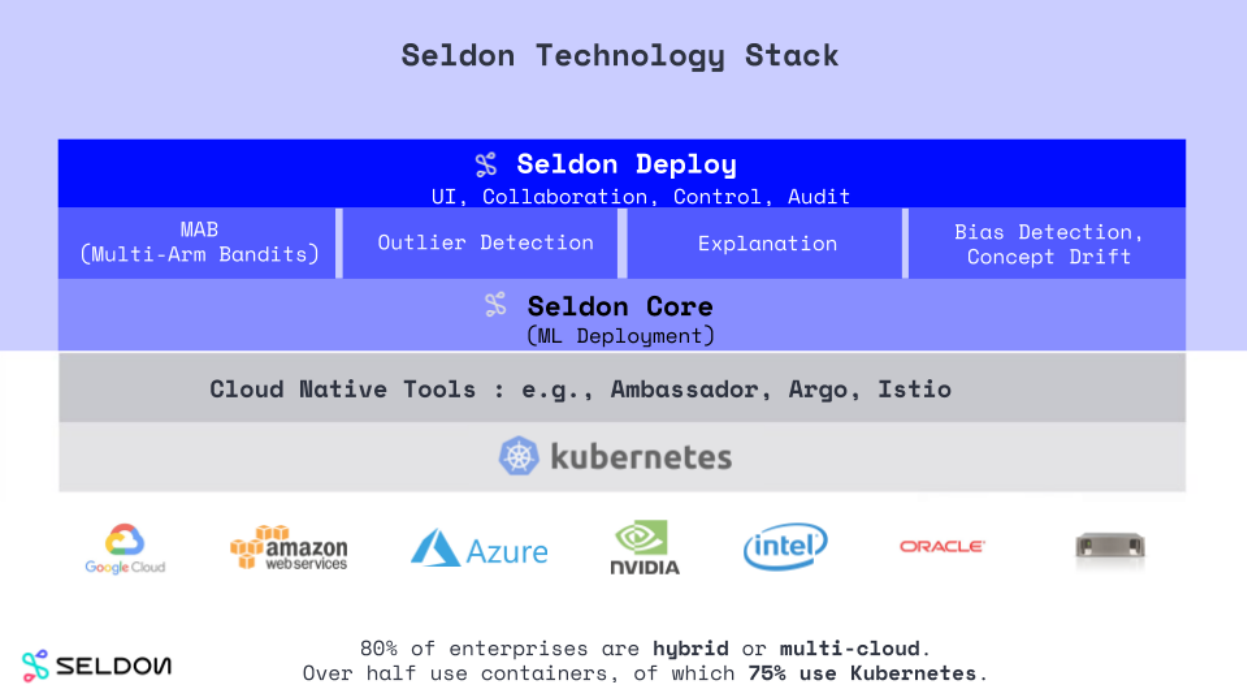
Seldon Deploy is an enterprise product to accelerate deployment management on top of the open source tools Seldon Core, KFServing and Seldon Alibi.

- Deploy machine learning models easily using industry leading Seldon and KfServing open projects.
- Ensure safe model deployment using the Gitops paradigm.
- Audit model predictions using Black Box Model Explainers.
- Monitor running models and search request/response logs.
- Update models via Canary workflows.

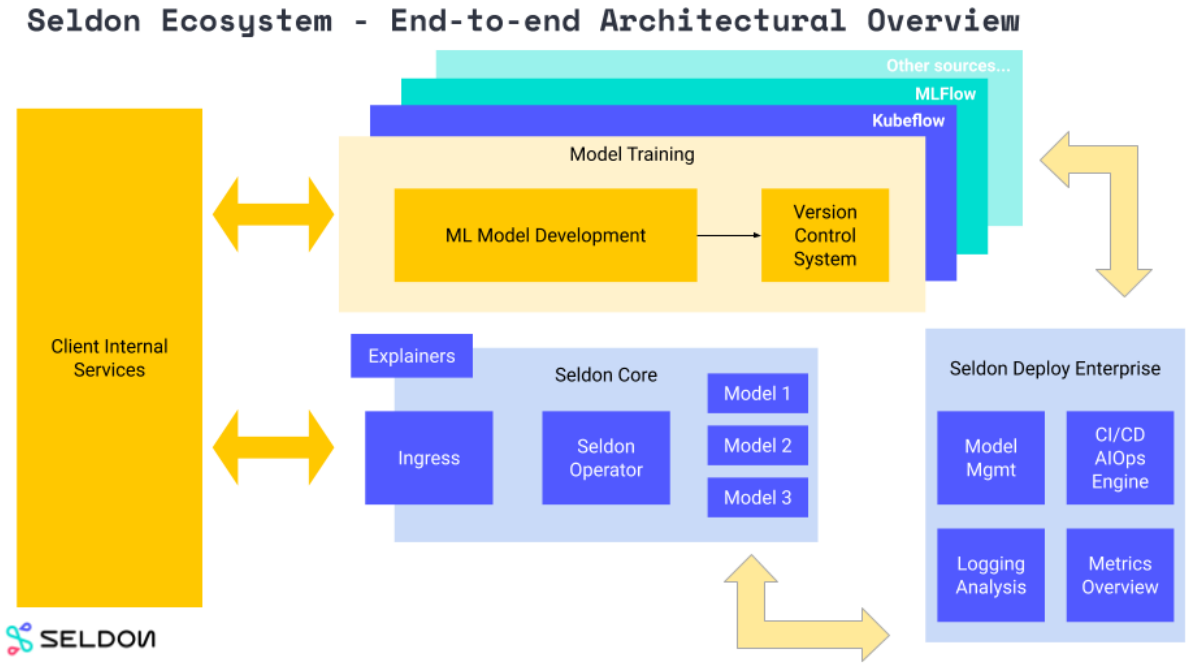
1.2 Seldon Core

Seldon Core is an open source platform for deploying machine learning models on a Kubernetes cluster.

- Deploy machine learning models in the cloud or on-premise.
- Get metrics and ensure proper governance and compliance for your running machine learning models.
- Create powerful inference graphs made up of multiple components.
- Provide a consistent serving layer for models built using heterogeneous ML toolkits.



Seldon Core fits into the stack alongside Seldon Deploy and your existing training pipelines as shown below:

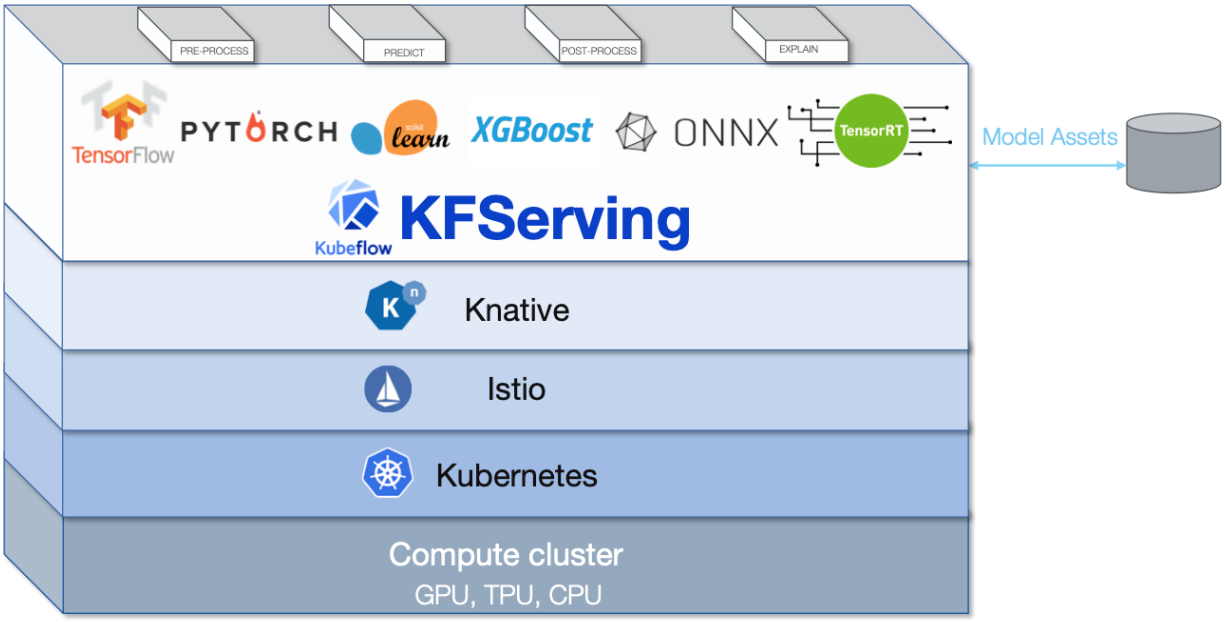


See the Seldon Core documentation for further details.

1.3 KFServing

KFServing provides a Kubernetes Custom Resource Definition for serving machine learning (ML) models on arbitrary frameworks. It aims to solve production model serving use cases by providing performant, high abstraction interfaces for common ML frameworks like Tensorflow, XGBoost, ScikitLearn, PyTorch, and ONNX.

It encapsulates the complexity of autoscaling, networking, health checking, and server configuration to bring cutting edge serving features like GPU Autoscaling, Scale to Zero, and Canary Rollouts to your ML deployments. It enables a simple, pluggable, and complete story for Production ML Serving including prediction, pre-processing, post-processing and explainability.



1.4 Alibi

Alibi is an open source Python library aimed at machine learning model inspection and interpretation. The initial focus on the library is on black-box, instance based model explanations.

Its goals are:

- Provide high quality reference implementations of black-box ML model explanation algorithms
- Define a consistent API for interpretable ML methods
- Support multiple use cases (e.g. tabular, text and image data classification, regression)
- Implement the latest model explanation, concept drift, algorithmic bias detection and other ML model monitoring and interpretation methods

---

1.4.1 Audience

1.4.1.1 Who should use Seldon Deploy?

There are various roles that should use Seldon Deploy.

Data Scientist

Why data scientists should use Seldon Deploy

Data scientist can use Seldon Deploy while they develop their models and when its is in Staging.

Some of the benefits include:

- Quickly deploy and update models.
- Understand model predictions through explanation techniques to help build better models.
- Ensure models are ready for the promotion to production.

Devops Engineer

Why Devops Engineers should use Seldon Deploy

Devops Engineers can use Seldon to ensure machine learning models are run safely and successfully in production.

Some of the benefits include:

- Deploy and rollout via Canaries new machine learning models.
- Ensure all actions performed on a model are auditable via GitOps.
- Monitor running models in real time.
- Provide feedback on particular model predictions.

Managers and Auditors

Why Managers and Auditors should use Seldon Deploy

Managers and Auditors can use Seldon to ensure machine learning models running in production are providing the benefits to the organisation and are properly audited.

Some of the benefits include:

- Check all models are running as expected.
- Audit actions there were performed on models.
- Provide feedback to clients and customer on particular model predictions that have been produced.

CHAPTER

TWO

RELEASE NOTES

Seldon Deploy Release Notes

2.1 Seldon Deploy 1.3.0

9 July 2021

2.1.1 What's New

- Feature Distributions Monitoring
- Policy Based Authorization (experimental)
- Deploying Models from Model Catalog
- Advanced query functionality for Model Metadata
- Prediction Schema support for Model Metadata
- Extended support for storage buckets using Rclone
- KFServing V2 protocol based explanations
- Env secret configuration for Seldon Deployments
- Support for client credentials grant auth using Seldon Deploy SDK

2.1.2 Bug fixes

- Fix for resource monitoring charts being duplicated for transformers and model containers
- Fix for invalid path when sending requests to MLServer deployments
- Fix for 50/50 canary traffic split causing Promote/Remove buttons to disappear
- Fix for long user group list causing overflow on About page

2.2 Seldon Deploy 1.2.1

3 June 2021

2.2.1 Bug fixes

- Fix for higher than expected CPU load when Model Catalogue is enabled.

2.3 Seldon Deploy 1.2.0

26 April 2021

2.3.1 Breaking Changes

- Moving detectors to user namespace (see upgrading)

2.3.2 What's New

- Model metadata store and catalogue
- Seldon Deploy server prometheus metrics
- Secrets for detector components
- Improved Gitops for kubernetes 1.18+
- Improved Triton support in wizard
- Oauth2 debugging documentation and testing tools

2.3.3 Bug fixes

- Fix for unexpected logout for group-restricted namespace
- Fix for SeldonDeployment without componentSpecs section not accessible in UI

2.4 Seldon Deploy 1.1.2

17 March 2021

2.4.1 Bug fixes

- Fix for outlier, drift and metrics server artefacts not being accessible from private buckets.

2.5 Seldon Deploy 1.1.0

10 February 2021

2.5.1 What's New

- Updated UI design
- Added both parameters and environment vars in wizard
- Added configurable runAsUser for detector components
- Added basic auth elastic creds
- Added active OpenAPI docs
- Added ability for not having to use clusterwide access

2.5.2 Bug fixes

- Fix for using input/output transformers
- Fix for needing ID Token Hint to the OIDC logout request

2.6 Seldon Deploy 1.0.0

27 November 2020

2.6.1 Breaking Changes

- Changes to the configuration (see upgrading)

2.6.2 What's New

- Drift detection analytics
- Batch inference processing
- Image explanations for Alibi explainer
- Seldon Core version 1.5.0 support
- kfserving version 0.4.1 support
- Kubeflow version 1.2 support
- OAuth annotations to Swagger spec
- Multitenancy support

2.6.3 Bug fixes

- Fix for metrics graph problems on Firefox

2.7 Seldon Deploy 0.9.0

6 November 2020

2.7.1 Breaking Changes

- Seldon Deploy now requires activation with a valid licence

2.7.2 What's New

- Usage Monitoring for user deployments
- Improved Outlier secrets handling
- Support for new versions of istio and knative
- Support for seldon core v1.4.0
- Data science metrics monitoring
- Improved outlier detection chart

2.7.3 Bug fixes

- Fix for not being able to filter predictor field for resource monitoring queries
- Fix for kfserving crd validation failures
- Fix for prometheus disk storage issues

2.8 Seldon Deploy 0.8.2

23 October 2020

2.8.1 Bug fixes

- YAML file extension support for Gitops
- jsonData requests to be displayed from the UI
- Updating request logger image (docker.io/seldonio/seldon-request-logger:1.3.0-json)
- Pod logs viewable for custom model containers

2.9 Seldon Deploy 0.8.1

28 September 2020

2.9.1 What's New

- Added secure flag for auth cookies on the browser

2.9.2 Bug fixes

- Fixed incorrect CPU/Memory metrics for additional predictors
- Supporting modified content-type headers behind reverse proxies

2.10 Seldon Deploy 0.8.0

8 September 2020

2.10.1 Breaking Changes

- The sd-install script is now called sd-install-default. Or for a kubeflow install there's sd-install-kubeflow.
- There are also helm chart values file changes. See upgrading section.

2.10.2 What's New

- Option to use SSH for git credentials via secret (see upgrading)
- New navigation sidebar
- Adding resource parameter option for token exchange to support ADFS/azure auth
- Adding resource_urn to auth code url
- Expose req logger replicas
- Adding theme customization for deploy UI (whitelabelling support)
- Reorganising sd install scripts
- Add mldeployments tag description to API spec
- Adding auth check for websocket events
- Adding logging to auth flow
- Updating seldon-core dependency to v1.2.3
- Updating swagger docs urls
- Add optional loadbalancersourceranges to helm chart
- Upgrading kfserving to v0.4.0 in default setup
- Adding support for SSH Key Access for Gitops

2.10.3 Bug fixes

- Adding fluentd labels to inferenceservice deployments for logs
- Filtering websocket data in context and updating namespaces selection
- Avoid user needing global git config

2.11 Seldon Deploy 0.7.0

7 July 2020

2.11.1 Breaking Changes

- There are some helm chart values file changes. See upgrading section.

2.11.2 What's New

- Support for openshift.
- Configurable request forms for installs with different cluster/ingress types (e.g. openshift).
- Remove/change file functionality during payload upload.
- Primary Request Logger in seldon-logs namespace.

2.11.3 Bug fixes

- Knative prometheus port conflict fixed.

2.12 Seldon Deploy 0.6.2

24 June 2020

2.12.1 Bug fixes

- App level auth redirect issue

2.13 Seldon Deploy 0.6.1

27 May 2020

2.13.1 Bug fixes

- Limit on auth cookie size
- Support token-based elastic auth
- Restructuring application paths and in-app navigation
- Install fails if kubeflow user and password missing
- Issue with env variables missing on canary promotion
- Resources prometheus can be different from requests prometheus

2.14 Seldon Deploy 0.6.0

22 April 2020

2.14.1 Breaking Changes

- New request logging update
- New gitops repository folder structure

2.14.2 What's New

- Outlier Detection Configuration and Monitoring
- App-level authentication with OIDC
- Tensorflow protocol for Seldon deployments
- MLFlow pre-packaged server in the deployment wizard
- Kubernetes pods and logging
- Displaying sync status for gitops namespaces
- New requests and resources monitoring dashboard
- Bitbucket support for gitops

2.14.3 Bug fixes

- Infinite loading if no valid labeled namespace was found
- No deployments were displayed if at least one was faulty

2.15 Seldon Deploy 0.5.2

17 March 2020

2.15.1 Bug fixes

- Browser timezone fix

CHAPTER

THREE

GETTING STARTED

Obtaining Access

To use Seldon Deploy for your enterprise, get in touch with Seldon for a free trial license *to apply*

3.1 Installation Types

- *Trial Installations* - Scripts that install default configurations that can be used to trial the Seldon Deploy cluster. These provide a simple way to get up and running to test the functionalities.
- *Production Installation* - Modular installation into existing cluster with instructions on how to leverage Seldon Deploy integrations. This is **recommended** installation for production setups.
- *Openshift Hat Installation* - Red Hat OpenShift installation.

See *Install Types* for more on how to choose.

3.2 Component integrations

Below are the integrations that Seldon Deploy supports, together with the versions currently supported for each of the component.

Component	Supported Versions	Integration level
seldon-core	1.9.1 and above	Required
Istio	1.7.3 and above	Required
Elasticsearch	6.x and above	Recommended
ArgoCD	1.4.x and above	Recommended
Prometheus	9.7 and above	Recommended
PostgreSQL (Model Metadata Storage)	9.6 and above	Recommended
Fluentd (or equivalent ELK log collection)	6.x and above	Optional
Dex (or equivalent OIDC provider - eg Keycloak)	2.x and above	Optional
KFserving	0.4.1 and above	Optional
Knative (Eventing, Serving & Monitoring)	0.18.x and above	Optional

Note KFServing requires istio 1.3.1. More on *istio/gateway options in architecture*. Detector components such as outliers requires Knative Eventing 0.18 (which requires k8s 1.16). Knative Serving 0.18. Any compatibility changes between versions get listed here and in the *upgrading section*

3.3 Pre-requisites

Dependency	Version	Notes
Docker-hub	n/a	https://hub.docker.com/ Access needs to be granted explicitly by Seldon in order to access resources such as docker images. A dockerhub account is needed and its credentials will be needed to setup imagePullSecrets.
Kubernetes	1.16 ... 1.18 recommended	Seldon Deploy is intended for Kubernetes version greater than or equal to 1.16 and less than or equal to 1.17.
Kubernetes	1.12 ... 1.19 available	See `Older Kubernetes Versions` and `Newer Kubernetes Versions` sections below for limitations.
Git Server	n/a	To use GitOps access an HTTPS access to git server is required. GitOps is optional, but suggested.
kubectl	within one minor version difference of your cluster	https://kubernetes.io/docs/tasks/tools/install-kubectl/
helm	3.0.0 or greater is required	https://helm.sh/docs/intro/install/

3.4 Cluster Requirements

Cloud Provider	Requirements	Notes
GKE	2 x e2-standard-8 nodes	This is 16vCPUs and 60GB RAM.
AWS	4 x t2.xlarge	This is to allow headroom to run models and because the full install includes kubeflow, knative, istio and the elastic and grafana-prometheus stacks.

Warning: Fewer and larger worker nodes are suggested as this requires less resource for the control plane. On EKS in particular increasing resources for the control plane has to be done explicitly. Too few nodes can also lead to problems on EKS with pods per node.

3.5 Older Kubernetes Versions

Knative 0.18+ requires Kubernetes 1.16. This places limitations on running Seldon Deploy on older Kubernetes versions.

Here is what can be done for older Kubernetes versions.

The request logger can be used with older knative versions. The helm values file contains this entry:

```
requestLogger:
  trigger:
    apiVersion: "eventing.knative.dev/v1"
```

Its value can be changed to "eventing.knative.dev/v1alpha1" for older versions of knative eventing.

The request logger can also be run without knative.

Both Seldon Core and KFServing have default URLs that their install instructions point to a knative broker in `seldon-logs` namespace (`executor.requestLogger.defaultEndpoint` for seldon core and `logger.defaultUrl` for KFServing). These can be changed to point to `seldon-request-logger.seldon-logs` (or your request logger location).

Detector component wizards and the Data Science metrics component wizard do need knative 0.18. For older k8s you can't run those demos.

3.6 Newer Kubernetes Versions

Seldon Deploy can work with kubernetes 1.19. At the time of writing KFServing cannot, which is an optional dependency.

3.7 References

3.7.1 Download resources

Download Seldon Deploy installation resources

```
TAG=1.3.0 && \
    docker create --name=tmp-sd-container seldonio/seldon-deploy-server:$TAG && \
    docker cp tmp-sd-container:/seldon-deploy-dist/seldon-deploy-install.tar.gz . && \
    docker rm -v tmp-sd-container

tar -xzf seldon-deploy-install.tar.gz
```

3.7.2 Applying a License

Applying a Seldon Deploy License

Obtaining Access

To use Seldon Deploy for your enterprise, get in touch with Seldon - request a free trial license via the request form.

A license can be entered in the UI when prompted.

Alternatively, it can be put in a text file called license and applied like this

```
kubectl create configmap -n seldon-system seldon-deploy-license --from-file=license -
↪o yaml --dry-run=client | kubectl apply -f -
kubectl delete pod -n seldon-system -l app.kubernetes.io/name=seldon-deploy || true
```

3.7.3 Trial Installation

Trial installations for setting up a new clusters

3.7.3.1 Trial Installation Options

These scripts will install the default configurations that can be used to trial the Seldon Deploy. These provide a simple way to get you up and running to test the functionalities that we offer.

- *Default Trial Installation*
- *Trial Installation with Kubeflow*
- *KIND Trial Installation - limited functionality*

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

3.7.3.2 Configuration

The trial installations of Seldon Deploy can be configured using a file called "sdconfig.txt".

The location and format of this file is

```
${HOME}/.config/seldon/seldon-deploy/sdconfig.txt
```

```
GIT_USER=<your GIT user>
GIT_TOKEN=<your GIT token>                          # This is either GitHub token or␣
↪Bitbucket App password
GIT_EMAIL=<your GIT email>
ENABLE_GIT_SSH_CREDS=false
SD_USER_EMAIL=<user email>                       # The user for access to Seldon Deploy␣
↪eg. admin@seldon.io
SD_PASSWORD=<user password>
EXTERNAL_HOST=                                      # This can be blank
EXTERNAL_PROTOCOL=https                             # "https" or "http"
KFSERVING_PROTOCOL=$EXTERNAL_PROTOCOL
MULTITENANT=false                                   # Restricts permissions to␣
↪namespace-level. Ask seldon before using.
ENABLE_APP_AUTH=true                                # 'true' or 'false'
VIRTUALSERVICE_CREATE=true
ENABLE_METADATA_POSTGRES=true                       # If false will skip installing␣
↪PostgreSQL in the trial cluster and metadata API will be unavailable.
ENABLE_OPA=false # If true will fetch OPA policies from a `seldon-deploy-policies`␣
↪config map from the namespace Deploy is running in.
ENABLE_PROJECT_BASED_AUTH=false # If true will authorize requests to resources based␣
↪on what access does the user have to all models in that resource.
```

Use the following script to check or create an initial file to fill in.

```
cd seldon-deploy-install
./check-config
```

3.7.3.3 References

Default Trial Installation

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

For AWS the EXTERNAL_PROTOCOL in the config file should be set to http. If https is needed on AWS then ssl should be configured on the istio gateway separately, not using the installer.

Simple Modular Installation

We will use an end-to-end installer which sets up Seldon Deploy and all ecosystem components.

These components include:

- Seldon Core
- Seldon Deploy
- Serverless Request Logging
- Grafana Visualisations
- GitOps Bitbucket/Github with ArgoCD

Install the Ecosystem

This first step will set up all ecosystem components for Seldon Deploy except for the GitOps ArgoCD component.

```
cd seldon-deploy-install
./prerequisites-setup/default-setup.sh
./sd-setup/sd-install-default
```

Hint: Ensure you have the correct **current context** set for your cluster, before starting the install.

Gitops

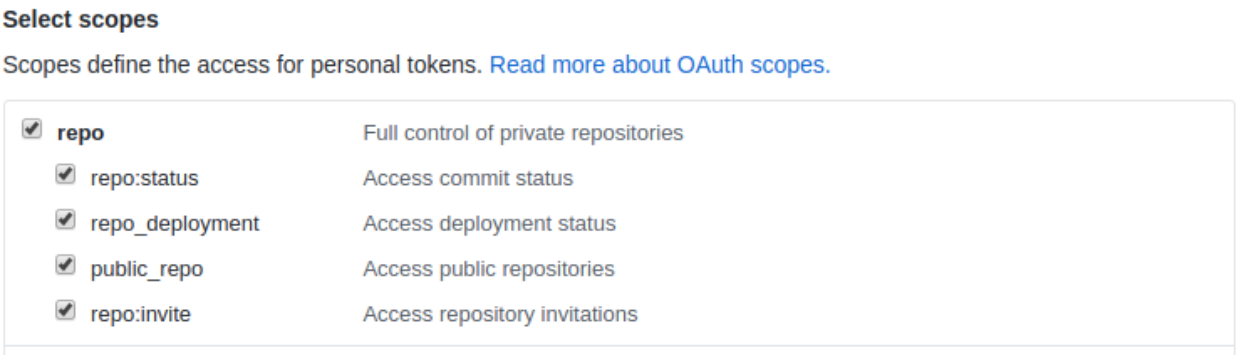
Gitops setup is optional. You will need an account on either github.com or bitbucket.org.

Installation requires `htpasswd`. Can be installed on Ubuntu, Debian and Mint with:

```
sudo apt install apache2-utils
```

GitHub

You will need personal access token configured with full control of private repositories as shown:

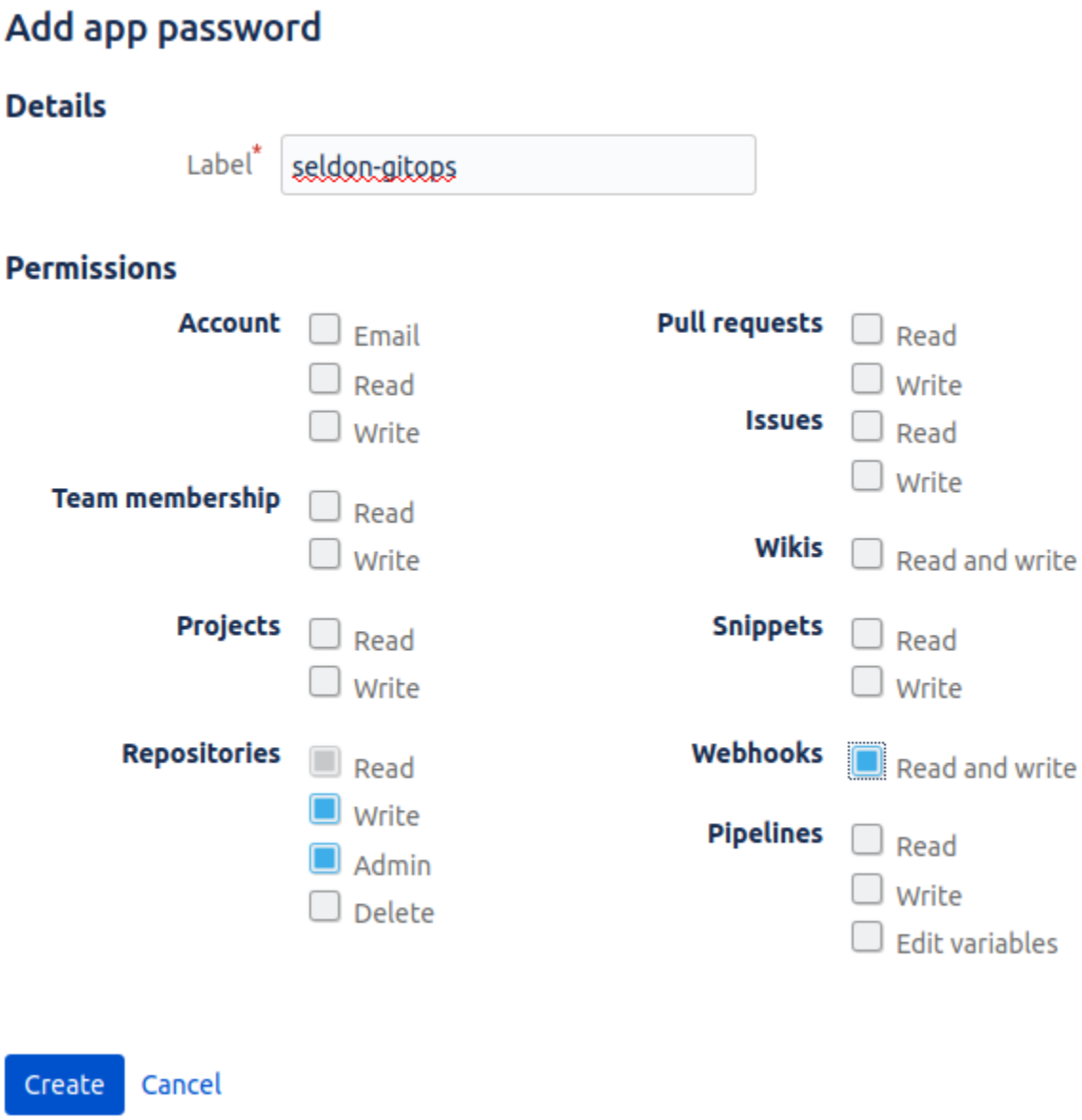


Then make sure that you setup `GIT_USER` and `GIT_TOKEN` in the `sdconfig.txt` file and run

```
./sd-setup/gitops-setup.sh seldon-gitops "seldon"
```

Bitbucket

You will need app password token configured with full control of private repositories as shown:



Then make sure that you setup `GIT_USER` and `GIT_TOKEN` in the `sdconfig.txt` file and run

```
GIT_PROVIDER=bitbucket ./sd-setup/gitops-setup.sh seldon-gitops "seldon"
```

If you prefer to install GitOps manually, please see *Bitbucket* for step by step instructions.

Open Policy Agent Authorization

To enable Open Policy Agent authorization the `ENABLE_OPA` variable must be set to `true` in the `sdconfig.txt` file. To use OPA to authorize projects as well, `ENABLE_PROJECT_BASED_AUTH` needs to also be `true`.

Accessing Seldon Deploy

After installation get url for the dashboard.

```
cd seldon-deploy-install
./sd-setup/show-seldon-deploy-url
```

Kubeflow Trial Installation

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

Simple Kubeflow Installation

Here we use an end-to-end kubeflow installer which sets up Seldon Deploy and all ecosystem components.

These components include:

- Seldon Core
- Seldon Deploy
- Serverless Request Logging
- Grafana Visualisations
- GitOps Bitbucket/Github with ArgoCD

Install the Ecosystem

The first step is to update the configuration options to use the kubeflow auth setup for your Seldon Deploy install as follows,

```
ENABLE_APP_AUTH=false
OIDC_PROVIDER=http://dex.auth.svc.cluster.local:5556/dex
```

The next step will set up all ecosystem components (including kubeflow) for Seldon Deploy except for the GitOps ArgoCD component.

```
cd seldon-deploy-install
./prerequisites-setup/kubeflow-setup.sh
./sd-setup/sd-install-kubeflow
```

Hint: Ensure you have the correct **current context** set for your cluster, before starting the install.

The kubeflow-setup.sh script is broken into sections so if you have an existing kubeflow install then its initial sections could be skipped.

The target kubeflow distribution is the istio_dex 1.2 distribution. The credentials for this are `admin@seldon.io/12341234`

Gitops

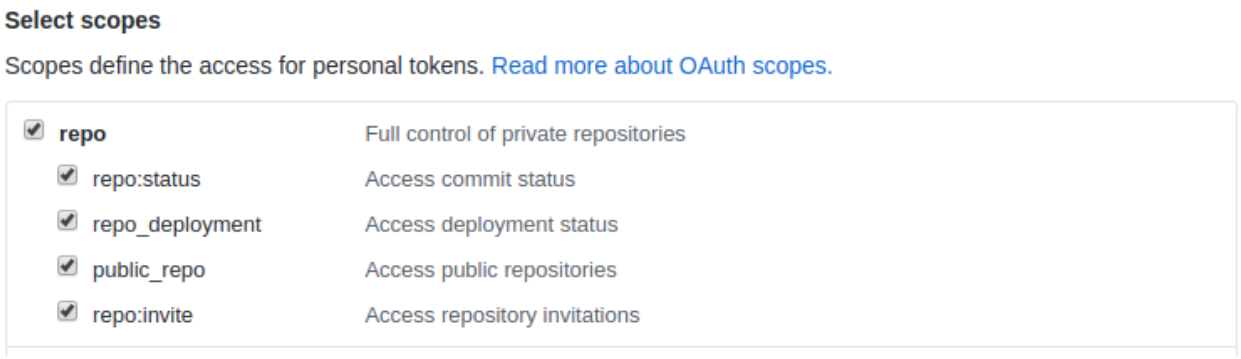
Gitops setup is optional. You will need an account on either github.com or bitbucket.org.

Installation requires `htpasswd`. Can be installed on Ubuntu, Debian and Mint with:

```
sudo apt install apache2-utils
```

GitHub

You will need personal access token configured with full control of private repositories as shown:

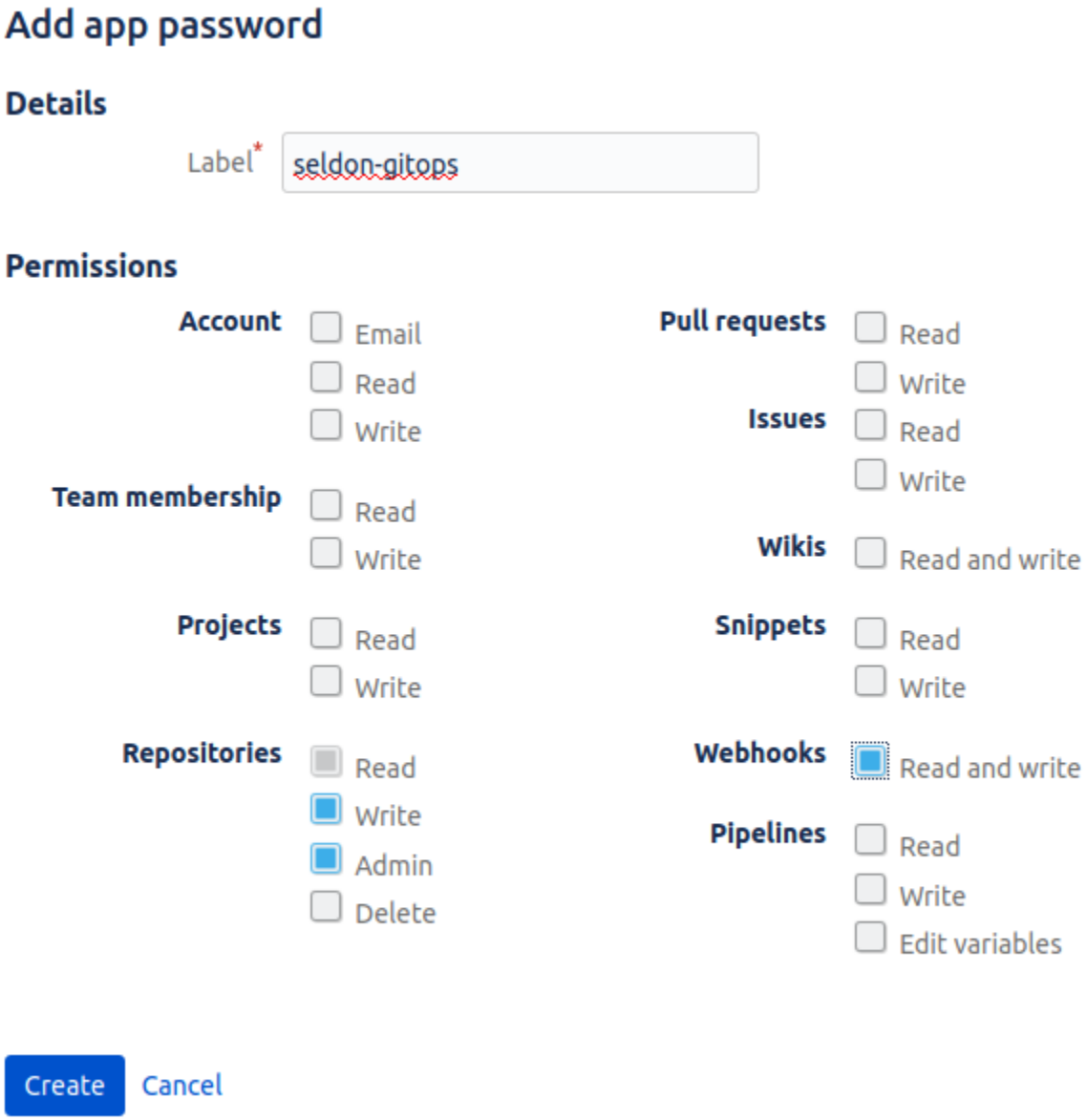


Then make sure that you setup `GIT_USER` and `GIT_TOKEN` in the `sdconfig.txt` file and run

```
./sd-setup/gitops-setup.sh seldon-gitops "default,kubeflow"
```

Bitbucket

You will need app password token configured with full control of private repositories as shown:



Then make sure that you setup `GIT_USER` and `GIT_TOKEN` in the `sdconfig.txt` file and run

```
GIT_PROVIDER=bitbucket ./sd-setup/gitops-setup.sh seldon-gitops "default,kubeflow"
```

If you prefer to install GitOps manually, please see *Bitbucket* for step by step instructions.

Accessing Seldon Deploy

After installation get url for the dashboard.

```
cd seldon-deploy-install
./sd-setup/show-seldon-deploy-url
```

The username and password will come from kubeflow, not Seldon config. If `kubeflow-setup.sh` is used then it will be the default for the istio_dex distro - `admin@kubeflow.org` and `12341234` (but the minio creds will be patched to match the seldon config).

You can double-check the user in `data.static_email` of `kubectl get cm -n auth dex-config -o yaml`

Kind Trial Installation

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all local tool pre-requisites are satisfied (e.g. helm cli) as well as KIND.

KIND Installation

Here we use an end-to-end installer to set up Seldon Deploy on KIND. Everything runs on the local machine with reduced resources compared to the cloud.

The components included are most of the functionality:

- Seldon Core
- Seldon Deploy
- Serverless Request Logging
- Grafana Visualisations

Components not included are:

- GitOps Bitbucket/Github with ArgoCD

This limitation is because it's not feasible for a GitHub webhook to call the local machine.

Install the Ecosystem

This step will set up a KIND cluster, install all ecosystem components for Seldon Deploy and Deploy itself.

```
cd seldon-deploy-install
./prerequisites-setup/kind-setup.sh
./sd-setup/sd-install-kind
```

Checking Configuration

If you setup the configuration file with `ENABLE_APP_AUTH` then either remove this or set it to `false`. `EXTERNAL_PROTOCOL` should be `http`.

Accessing Seldon Deploy

On KIND we need to port-forward in order to access the application. In a terminal run the below and keep it running:

```
kubectl port-forward -n istio-system svc/istio-ingressgateway 8080:80
```

Then open a browser at `http://localhost:8080/seldon-deploy/`.

3.7.4 Production Installation

Custom installation crafted to your needs

Modular installation that allows for fully configurable installation of Seldon Deploy.

3.7.4.1 Components overview

Required components:

- Seldon Core
- Seldon Deploy
- *Istio Ingress*

Optional components:

- *Metrics Monitoring with Prometheus*
- *Request Logging with Elasticsearch*
- *GitOps setup with ArgoCD*
- *App Level Authentication with OIDC provider*
- *Minio*
- *Argo*
- *PostgreSQL*

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

3.7.4.2 References

Seldon Deploy

This page contains instructions and details for the advanced instalallation of Seldon Deploy and Seldon Core.

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

Prepare seldon namespaces

We start with creating a namespace that will hold Seldon's main components, traditionally naming it `seldon-system`.

```
kubectl create ns seldon-system || echo "Namespace seldon-system already exists"
```

Also a namespace for any components related to request logging.

```
kubectl create ns seldon-logs || echo "Namespace seldon-logs already exists"
```

Seldon Deploy (installation)

To install a vanilla version of Seldon Deploy:

```
helm upgrade seldon-deploy ./seldon-deploy-install/sd-setup/helm-charts/seldon-deploy/
↪ \
    --set image.image=seldonio/seldon-deploy-server:1.3.0 \
    --set virtualService.create=false \
    --set requestLogger.create=false \
    --set gitops.argocd.enabled=false \
    --set elasticsearch.basicAuth=false \
    --set enableAppAuth=false \
    --namespace=seldon-system \
    --install
```

Now we can just wait until the Seldon Deploy roll-out finishes

```
kubectl rollout status deployment/seldon-deploy -n seldon-system
```

Note that due the flags set above this vanilla version of Seldon Deploy would lack certain features such as request logging and gitops.

Seldon Deploy (configuration)

Seldon Deploy is installed using helm-charts. It is recommended to use helm `upgrade` command with `--install` flag. This will allow to re-issue the command to bring new settings or upgrade the Seldon Deploy version.

Installation options can be set in two ways:

- Using yaml values file specified with helm `-f` flag (recommended)
- Using helm `--set` flags

Please, see `helm upgrade --help` and official documentation for more information.

Using a yaml file with values is recommended as it provides a more reproducible installation.

The default installation values are defined in the helm-charts and can be viewed with

```
cat seldon-deploy-install/sd-setup/helm-charts/seldon-deploy/values.yaml
```

You can view values set on your current installation with

```
helm get values -a -n seldon-system seldon-deploy
```

Note that `-a` flag tells `helm` to give you all values - skipping that flag you will only see values set by you explicitly on previous installation.

Following `deploy-values.yaml` file is equivalent to presented above `--set` options

```
image:
  image: seldonio/seldon-deploy-server:1.3.0

rbac:
  clusterWide: true

virtualService:
  create: false

requestLogger:
  create: false

gitops:
  argocd:
    enabled: false

enableAppAuth: false

elasticsearch:
  basicAuth: false
```

You can install Seldon Deploy using the above values file with following command

```
helm upgrade seldon-deploy ./seldon-deploy-install/sd-setup/helm-charts/seldon-deploy/
↪ \
    -f deploy-values.yaml \
    --namespace=seldon-system \
    --install
```

Seldon Core (installation)

Seldon Core can be installed using published helm-charts. To add helm charts we can use following script

```
helm repo add seldonio https://storage.googleapis.com/seldon-charts
helm repo update
```

You can now install Seldon Core without any integrations with the script below. Make sure to follow relevant individual feature workflow installation sections to enable specific features. For further detail on each of the parameters available in the installation script you can read the helm chart values section in the seldon core documentation.

```
helm upgrade seldon-core seldonio/seldon-core-operator \
    --version 1.9.1 \
    --namespace seldon-system \
    --install
```

Seldon Core (configuration)

Similarly to the Seldon Deploy case we will assume that there is a `core-values.yaml` file specifying Seldon Core related configurations. The helm command will now read

```
helm upgrade seldon-core seldonio/seldon-core-operator \
    -f core-values.yaml \
    --version 1.9.1 \
    --namespace seldon-system \
    --install
```

Ensuring visibility on namespaces

The `seldon.restricted=false` label is required on namespaces accessible by Seldon Deploy. If you don't add this annotation to the namespace, you will not be able to see it in the UI.

```
kubectl create namespace seldon || echo "namespace seldon exists"
kubectl label ns seldon seldon.restricted=false --overwrite=true
```

For more information see our *Namespace Visibility* documentation.

Istio Ingress

Configure Seldon ingress with Istio

Istio is a production-ready service mesh solution, which Seldon Deploy uses for routing of traffic. Similarly, traffic can also be managed by Istio for dynamically created seldon models, which simplifies how users can send requests to a single endpoint.

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components are installed.

You can use the istio CLI to install the istio containers into your cluster. Istio provides multiple installation configuration profiles, it is important to choose the most appropriate configuration, especially as Istio is often used for

applications beyond just Seldon itself. You can read more about the profiles and configurations in the Istio documentation.

Installing Istio Pods

First, obtain `istioctl` CLI that will help in installation of Istio. We will be installing version 1.7.3 which is compatible with kubernetes clusters version 1.16-1.18.

```
export ISTIO_VERSION=1.7.3
curl -L https://istio.io/downloadIstio | ISTIO_VERSION=${ISTIO_VERSION} sh -
```

Once you have downloaded the command line interface, we can configure a cluster local gateway and other istio components for our installation with:

```
cat << EOF > ./local-cluster-gateway.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      proxy:
        autoInject: disabled
      useMCP: false
  addonComponents:
    pilot:
      enabled: true
    prometheus:
      enabled: false
  components:
    ingressGateways:
      - name: cluster-local-gateway
        enabled: true
        label:
          istio: cluster-local-gateway
          app: cluster-local-gateway
        k8s:
          service:
            type: ClusterIP
            ports:
            - port: 15020
              name: status-port
            - port: 80
              targetPort: 8080
              name: http2
            - port: 443
              targetPort: 8443
              name: https
  values:
    gateways:
      istio-ingressgateway:
        debug: error
EOF
```

The installation can then be performed with:

```
./istio-1.7.3/bin/istioctl install -f local-cluster-gateway.yaml
```

It's worth emphasising that Istio in itself is quite a feature-rich open source project, which is often leveraged as central service mesh for clusters. If the use-cases are more specific, there are quite a lot of different configurations that can be chosen, examples covered above include the different installation profiles, but also there are instruction like the documentation in the istio section for KNative.

Installing Seldon with Istio Configuration Enabled

In order to install Seldon with Istio enabled there are a couple of requirements, namely:

- An Istio gateway needs to be created
- Seldon Core Helm Chart needs to be installed with the istio values
- Seldon Deploy Helm Chart needs to be installed with istio values

Setting up Istio Gateway

Istio Gateway is required to access deployments and Seldon Deploy itself

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: seldon-gateway
  namespace: istio-system
spec:
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  servers:
    - hosts:
        - "*"
      port:
        name: http
        number: 80
        protocol: HTTP
```

Example of https-enabled gateway can be found under

```
seldon-deploy-install/prerequisites-setup/istio/seldon-gateway.yaml
```

The Istio gateway can be created in the Istio namespace, but it can also be created in a different namespace, you will need to make sure that the configuration of Seldon Deploy and Core are also aligned to the namespace and gateway name.

Install Seldon Core with Istio Enabled

You can now add the following values in your `core-values.yaml` file. You need to make sure that the value for the gateway is `<namespace>/<gatewayname><namespace>`.

```
istio:
  enabled: true
  gateway: "istio-system/seldon-gateway"
```

Install Seldon Deploy with Istio Enabled

Similarly you would add following entries to your `deploy-values.yaml` file

```
ingressGateway:
  seldonIngressService: "istio-ingressgateway"
  kfServingIngressService: "istio-ingressgateway"
  ingressNamespace: "istio-system"

virtualService:
  create: true
  gateways:
    - istio-system/seldon-gateway
```

Find address of your Seldon Deploy

Following script can help you find address of your Seldon Deploy instance running with Istio:

```
ISTIO_INGRESS=$(kubectl get svc -n istio-system istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}')
ISTIO_INGRESS+=$(kubectl get svc -n istio-system istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].hostname}')

echo "Seldon Deploy: http://$ISTIO_INGRESS/seldon-deploy/"
```

Metrics Monitoring

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components are installed.

Installation

The analytics component is configured with the Prometheus integration. The monitoring for Seldon Deploy is based on the Open Source Analytics package which brings in Prometheus (and Grafana) and is required for metrics collection.

Before installing we should set up a recording rules file. Name this `model-usage.rules.yml`. The contents of this file are given in the last section.

Copy the default `model-usage.rules.yml` (and edit if desired)

```
cp ./seldon-deploy-install/prerequisites-setup/prometheus/model-usage.rules.yml model-
↪usage.rules.yml
```

Create a configmap from the file with

```
kubectl create configmap -n seldon-system model-usage-rules --from-file=model-usage.
↪rules.yml --dry-run=client -o yaml | kubectl apply -f -
```

Without these usage rules you may see warnings about usage.

This can be mounted by setting the below `extraConfigmapMounts` in an analytics-values.yaml:

```
cat << EOF > ./analytics-values.yaml
grafana:
  resources:
    limits:
      cpu: 200m
      memory: 220Mi
    requests:
      cpu: 50m
      memory: 110Mi

prometheus:
  alertmanager:
    resources:
      limits:
        cpu: 50m
        memory: 64Mi
      requests:
        cpu: 10m
        memory: 32Mi
  nodeExporter:
    service:
      hostPort: 9200
      servicePort: 9200
    resources:
      limits:
        cpu: 200m
        memory: 220Mi
      requests:
        cpu: 50m
        memory: 110Mi
  server:
    livenessProbePeriodSeconds: 30
    retention: "90d"
    extraArgs:
      query.max-concurrency: 400
      storage.remote.read-concurrent-limit: 30
    persistentVolume:
      enabled: true
      existingClaim: ""
      mountPath: /data
      size: 32Gi
    resources:
      limits:
        cpu: 2
        memory: 4Gi
      requests:
        cpu: 800m
        memory: 1Gi
    extraConfigmapMounts:
      - name: prometheus-config-volume
        mountPath: /etc/prometheus/conf/
        subPath: ""
        configMap: prometheus-server-conf
        readOnly: true
```

(continues on next page)

(continued from previous page)

```
      - name: prometheus-rules-volume
        mountPath: /etc/prometheus-rules
        subPath: ""
        configMap: prometheus-rules
        readOnly: true
      - name: model-usage-rules-volume
        mountPath: /etc/prometheus-rules/model-usage/
        subPath: ""
        configMap: model-usage-rules
        readOnly: true
EOF
```

Other settings in the above are suggested only. Configure to suit your disk availability.

```
helm repo add seldonio https://storage.googleapis.com/seldon-charts
helm repo update

helm upgrade seldon-core-analytics seldonio/seldon-core-analytics \
    --version 1.9.1 \
    --namespace seldon-system \
    -f analytics-values.yaml \
    --install
```

This Prometheus installation is already configured to scrape metrics from Seldon Deployments. Seldon Core documentation on analytics covers metrics discussion and configuration of Prometheus itself.

It's possible to leverage further custom parameters provided by the helm charts, such as:

- grafana_prom_admin_password - The admin password for grafana to use
- persistence.enabled - This provides the configuration to enable prometheus persistence

Bringing your own Prometheus

It is possible to use your own Prometheus instance - see `prometheus` section in the default values file

```
seldon-deploy-install/sd-setup/helm-charts/seldon-deploy/values.yaml
```

Verification / Troubleshooting

We can port-forward prometheus in order to check it. With seldon-core-analytics the prometheus service we can do this with:

```
kubectl port-forward -n seldon-system svc/seldon-core-analytics-prometheus-seldon␣
↪9090:80
```

Then go to `localhost:9090` in the browser.

To confirm the recording rules are present, go to `Status > Rules` and search for `model-usage`.

If you have a seldon model running, go to `Status > Targets` and search for `seldon_app` or just `seldon`. Any targets for seldon models should be green.

On the `/graph` page if you select from the `insert metric at cursor` drop-down there should be metrics that begin with the names seldon.

Request Logging

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components are installed.

Install KNative Eventing

Knative Eventing is used for request logging. See *Knative section* for steps.

Knative eventing is recommended but see `Running without KNative` below if this represents a major barrier for you.

Install EFK Stack

We suggest using the *OpenDistro* version of elastic for a trial and provide *instructions*.

Other *flavours* are also available.

Fluentd is used for collecting application logs. We use the request logger component for storing the bodies of http requests to models and other data about those, as elastic documents.

Seldon Core and Deploy Configuration

For Seldon Core add to your `core-values.yaml` following options

```
executor:
  requestLogger:
    defaultEndpoint: "http://broker-ingress.knative-eventing.svc.cluster.local/seldon-
↪logs/default"
```

For Seldon Deploy add to your `deploy-values.yaml` following options

```
requestLogger:
  create: true
```

If using opendistro, it should also have auth enabled, in line with the instructions in that section (referenced above).

Using Deploy Metadata - Auth from Request Logger to Deploy

The request logger can optionally obtain a prediction schema from Seldon Deploy Metadata service if enabled. This can be used to enrich prediction logs (e.g. add category name for categorical features and identify probabilistic features) for better logging and advanced monitoring.

To enable this feature, it is necessary to ensure that the Seldon Deploy instance needs to be configured with a valid licence. And then follow the steps below,

1. Ensure model metadata storage feature is enabled i.e `metadata.pg.enabled` is set to true. For more details see postgres setup.

2. Have an auth provider that supports client credentials grant flow and enable this on a client. This can also be done using a password grant flow for other cases where a client credentials flow is not supported.
3. Create an auth secret for the request logger configuration as below, setting the parameters as per your environment.

```
kubectl create secret generic {request-logger-auth-secret-name} -n seldon-logs \
  --from-literal=OIDC_PROVIDER="${OIDC_PROVIDER}" \
  --from-literal=CLIENT_ID="${CLIENT_ID}" \
  --from-literal=CLIENT_SECRET="${CLIENT_SECRET}" \
  --from-literal=OIDC_AUTH_METHOD="client_credentials" \
  --from-literal=OIDC_SCOPES="${OIDC_SCOPES}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

4. Ensure `requestLogger.deployHost` and `requestLogger.authSecret` are set. Default helm chart values should be fine.

```
requestLogger:
  authSecret: {request-logger-auth-secret-name}
```

Trial Installation Setup

In the provided *trial installation*, a keycloak client is configured called 'sd-api' and a service account is also created. The scripts are under `./prerequisites-setup/keycloak`. The *trial installation* scripts create the auth secret needed and fully configures the request logger and metadata automatically. But if you are installing components yourself then you need to set this up specifically. Setting up a separate client is optional as the main deploy client could be used. In keycloak, setting up a service account is required for client credentials auth flow.

Custom Request Logger

It's possible for you to add your own custom request logger, with any custom logic you'd like to add. In order to do this, you need to make sure each Seldon Deployment points to the endpoint of your custom request logger.

For this, you will need to make sure you enable the Seldon Operator Environment variable for the logger. Prior to v1.2.x this was `"EXECUTOR_REQUEST_LOGGER_DEFAULT_ENDPOINT_PREFIX"` and then became `"EXECUTOR_REQUEST_LOGGER_DEFAULT_ENDPOINT"`. It can be enabled through the core helm chart's `executor.requestLogger.defaultEndpoint`.

Below we show an example of how you would do this for our non-knative default request logger

Running without KNative

It's also possible to set up request logging without KNative dependency. For this, you will have to run a non-knative request logger, which you can trigger by running the configuration below.

Make sure that you edit the elasticsearch host variable below to point to the correct elasticsearch service address.

Important: for a normal install use the helm chart to install the logger. But that uses knative eventing. If you don't want that then you can refer to the helm chart to create your own logger spec like the below.

To do this, first disable the `requestLogger` that is installed by the helm charts by setting in the `deploy-values.yaml`

```
requestLogger:
  create: false
```

then create the deployment with your custom logger:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: seldon-request-logger
  namespace: seldon-logs
  labels:
    app: seldon-request-logger
spec:
  replicas: 2
  selector:
    matchLabels:
      app: seldon-request-logger
  template:
    metadata:
      labels:
        app: seldon-request-logger
    spec:
      containers:
        - name: user-container
          image: docker.io/seldonio/seldon-request-logger:1.9.1
          imagePullPolicy: Always
          env:
            - name: ELASTICSEARCH_HOST
              value: "elasticsearch-opendistro-es-client-service.seldon-logs.svc.
↪cluster.local"
            - name: ELASTICSEARCH_PORT
              value: "9200"
            - name: ELASTICSEARCH_PROTOCOL
              value: "https"
            - name: ELASTICSEARCH_USER
              valueFrom:
                secretKeyRef:
                  name: elastic-credentials
                  key: username
            - name: ELASTICSEARCH_PASS
              valueFrom:
                secretKeyRef:
                  name: elastic-credentials
                  key: password
---
apiVersion: v1
kind: Service
metadata:
  name: seldon-request-logger
  namespace: seldon-logs
spec:
  selector:
    app: seldon-request-logger
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

For a secured elastic you need to set `ELASTICSEARCH_PROTOCOL`, `ELASTICSEARCH_USER` and `ELASTICSEARCH_PASS` that in above example are sourced from the `elastic-credentials` secret.

Configure Seldon Core to use that endpoint

In order to make sure the Seldon Deployments are sending the requests to that endpoint, you will need to make sure you provide the request logger prefix. In this case you will need the following extra attributes in the Seldon Core `values.yaml`:

```
executor:
  requestLogger:
    defaultEndpoint: "http://seldon-request-logger.seldon-logs.svc.cluster.local"
```

It's important that you make sure it's on that format, which is `http://<LOGGER_SERVICE>.<LOGGER_NAMESPACE>.svc.cluster.local`.

If you prefer to have one request logger per kubernetes namespace set `defaultEndpoint: http://<LOGGER_SERVICE>.` (note the trailing dot) - the Seldon Service Orchestrator will then add the namespace where the Seldon Deployment is running as a suffix.

Overriding Request Logger Endpoint for specific Seldon Deployment

Once you have created the request logger, now you have to make sure your deployments are pointing to the correct custom request logger. You can set up the custom request logger address by adding the following configuration to every Seldon Core SeldonDeployment file:

```
      logger:
        mode: all
        url: http://seldon-request-logger.default
```

The `mode` configuration can be set to `request`, `response` or `all`.

The `url` is where this should be logged to. There's a similar example for KFServing.

Authentication on Elasticsearch

The Seldon Deploy helm values file has two options for connecting to a secured elastic.

One is token-based authentication. Use this if you have an auth token. This is used for openshift cluster logging flavour of elastic.

The other option is basic authentication.

Elastic can be configured with basic auth. Note this requires an xpack feature.

Similar then needs to be applied for kibana - note the env vars are not quite the same.

And fluentd.

For Deploy this would need secrets in the namespaces seldon-logs (containing elastic and req logger) and seldon-system (containing Deploy) as Deploy would need to speak to elastic using the secret.

This could look like:

```
ELASTIC_USER=admin
ELASTIC_PASSWORD=admin

kubectl create secret generic elastic-credentials -n seldon-logs \
  --from-literal=username="${ELASTIC_USER}" \
  --from-literal=password="${ELASTIC_PASSWORD}" \
```

(continues on next page)

(continued from previous page)

```
  --dry-run=client -o yaml | kubectl apply -f -

kubectl create secret generic elastic-credentials -n seldon-system \
  --from-literal=username="${ELASTIC_USER}" \
  --from-literal=password="${ELASTIC_PASSWORD}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

Debugging

Often issues with request logging turn out to be *knative* or *elasticsearch* issues. Start by checking *knative*.

Knative Installation

This section walks through installation of knative for Seldon Deploy.

Knative eventing and serving is used for request logging and for post-predict detector components (outlier, drift, metrics). More in *request logging*

Knative serving is also needed for *KFServing*.

Knative can be installed in other ways - if you have an existing install then just apply steps that customize.

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components and *Istio* are installed.

Install KNative Serving

Install `KNATIVE Serving`

```
KNATIVE_SERVING_URL=https://github.com/knative/serving/releases/download
SERVING_VERSION=v0.18.1
SERVING_BASE_VERSON=v0.18.0

kubectl apply -f ${KNATIVE_SERVING_URL}/${SERVING_VERSION}/serving-crds.yaml
kubectl apply -f ${KNATIVE_SERVING_URL}/${SERVING_VERSION}/serving-core.yaml

kubectl apply -f https://github.com/knative-sandbox/net-istio/releases/download/$
↪{SERVING_BASE_VERSON}/release.yaml
```

If you are using seldon core analytics for prometheus (recommended) then for knative metrics add these annotations:

```
kubectl annotate -n knative-serving service autoscaler prometheus.io/scrape=true
kubectl annotate -n knative-serving service autoscaler prometheus.io/port=9090

kubectl annotate -n knative-serving service activator-service prometheus.io/
↪scrape=true
kubectl annotate -n knative-serving service activator-service prometheus.io/port=9090
```

Configure Cluster Local Gateway

KNative requires a Cluster Local Gateway to work properly. This can be added to an existing Istio 1.6.x installation by generating required manifests

```
cat << EOF > ./local-cluster-gateway.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  profile: empty
  components:
    ingressGateways:
      - name: cluster-local-gateway
        enabled: true
        label:
          istio: cluster-local-gateway
          app: cluster-local-gateway
        k8s:
          service:
            type: ClusterIP
            ports:
            - port: 15020
              name: status-port
            - port: 80
              name: http2
            - port: 443
              name: https
  values:
    gateways:
      istio-ingressgateway:
        debug: error
EOF

istioctl manifest generate -f local-cluster-gateway.yaml > manifest.yaml
```

Note the `profile: empty` line. This ensures that generated manifest only contain gateway related resources.

Once manifest is generated, inspect them, and then use `kubectl` to apply them

```
kubectl apply -f manifest.yaml
```

The above manifest and `istioctl` command no longer works with istioctl 1.7. For 1.7 we suggest installing istio with

```
cat << EOF > ./local-cluster-gateway.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      proxy:
        autoInject: disabled
      useMCP: false
  addonComponents:
    pilot:
      enabled: true
    prometheus:
      enabled: false
```

(continues on next page)

(continued from previous page)

```
  components:
    ingressGateways:
      - name: cluster-local-gateway
        enabled: true
        label:
          istio: cluster-local-gateway
          app: cluster-local-gateway
        k8s:
          service:
            type: ClusterIP
            ports:
            - port: 15020
              name: status-port
            - port: 80
              targetPort: 8080
              name: http2
            - port: 443
              targetPort: 8443
              name: https
  values:
    gateways:
      istio-ingressgateway:
        debug: error
EOF

./istioctl install -f local-cluster-gateway.yaml
```

Read more about gateway configuration in Istio's documentation.

Test Knative Serving

To check installed version of KNative Serving:

```
kubectl get namespace knative-serving -o 'go-template={{index .metadata.labels
↪"serving.knative.dev/release"}}'
```

To verify the install, `kubectl apply -f` on a file containing the below:

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: helloworld-go
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: gcr.io/knative-samples/helloworld-go
          env:
            - name: TARGET
              value: "Go Sample v1"
```

If this is applied in the default namespace then do a pod watch in another window (`kubectl get pod -n default -w`) curl it with below (else change `default` to chosen namespace):

```
kubectl run --quiet=true -it --rm curl --image=radial/busyboxplus:curl --
↪restart=Never --  \
curl -v -X GET "http://helloworld-go.default.svc.cluster.local"
```

You should get a successful response and also a pod should come up in default. If you don't then see note below on private registries before looking at resources such as seldon and knative slack.

Clean up with `kubectl delete -f` on the same file as before.

Install KNative Eventing

```
KNATIVE_EVENTING_URL=https://github.com/knative/eventing/releases/download
EVENTING_VERSION=v0.18.3

kubectl apply --filename ${KNATIVE_EVENTING_URL}/${EVENTING_VERSION}/eventing-crds.
↪yaml
kubectl apply --filename ${KNATIVE_EVENTING_URL}/${EVENTING_VERSION}/eventing-core.
↪yaml

kubectl apply --filename ${KNATIVE_EVENTING_URL}/${EVENTING_VERSION}/in-memory-
↪channel.yaml
kubectl apply --filename ${KNATIVE_EVENTING_URL}/${EVENTING_VERSION}/mt-channel-
↪broker.yaml

kubectl rollout status -n knative-eventing deployment/imc-controller
```

Configure KNative Eventing broker

Create knative event broker that will handle the logging.

```
kubectl create -f - <<EOF
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  name: default
  namespace: seldon-logs
EOF
```

Test KNative Eventing

To check installed version of KNative Eventing

```
kubectl get namespace knative-eventing -o 'go-template={{index .metadata.labels
↪"eventing.knative.dev/release"}}'
```

To test knative eventing it is easiest to have Seldon fully installed with request logging and a model running.

Make a prediction to a model following one of the *seldon core demos*. You should see entries under `Requests`.

If you see entries under requests, you are all good.

If you don't see entries under requests, first find the request logger pod in the seldon-logs namespace. Tail its logs (`kubectl logs -n seldon-logs <pod-name> -f`) and make a request again. Do you see output?

If this doesn't work then find the pod for the model representing the SeldonDeployment. Tail the logs of the seldon-container-engine container and make a prediction again.

If the predictions aren't sending then it could be a problem with the broker url `executor.requestLogger.defaultEndpoint` in `helm get values -n seldon-system seldon-core`) or the broker (`kubectl get broker -n seldon-logs`).

If there's no requests and no obvious problem with the broker transmission then it could be the trigger stage.

First try `kubectl get trigger -n seldon-logs` to check the trigger status.

If that looks healthy then we need to debug the knative trigger process.

Do a `kubectl apply -f` to the default namespace on a file containing the below (or change references to `default` for a different namespace):

```
# event-display app deploment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-display
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels: &labels
      app: event-display
  template:
    metadata:
      labels: *labels
    spec:
      containers:
        - name: helloworld-python
          image: gcr.io/knative-releases/github.com/knative/eventing-sources/cmd/
↪event_display
---
# Service that exposes event-display app.
# This will be the subscriber for the Trigger
kind: Service
apiVersion: v1
metadata:
  name: event-display
  namespace: default
spec:
  selector:
    app: event-display
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
---
# Trigger to send events to service above
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: event-display
  namespace: seldon-logs
spec:
  broker: default
```

(continues on next page)

(continued from previous page)

```
  subscriber:
    uri: http://event-display.default:80
```

Now find the event-display pod and tail its logs (`kubectl get pod -n default` and `kubectl logs <pod_name>`). Make a prediction to a model following one of the *seldon core demos*.

You should see something in the event-display logs. Even an event decoding error message is good.

To eliminate any seldon components, we can send an event directly to the broker. There is an example in the knative docs

What we've done now corresponds to the knative eventing hello-world. Nothing at all in the event-display pod means knative eventing is not working.

Occasionally you see a `RevisionMissing` status on the `ksvc` and a `ContainerCreating` message on its `Revision`. If this happens check the `Deployment` and if no issues then delete and try again.

Hopefully you've got things working before here. If not then check the pods in the `knative-eventing` namespace. If that doesn't help find the problem then the knative slack and/or seldon slack can help with further debugging.

Knative with a private registry

By default knative assumes the image registries used for images will be public. If you use a company-internal private one, then you have to configure that.

There is a property called queueSidecarImage in the config-deployment configmap in the knative-serving namespace. This needs to be edited to point to your registry

Tag resolution confiuration can also be required. We suggest to try the above first. The knative docs provide more details on tag resolution

Elasticsearch Installation

Installing flavours of elasticsearch.

If you do not already have elasticsearch then we suggest to use the OpenDistro version of elasticsearch.

If you prefer the version of elasticsearch distributed by elastic then instructions for this are also below.

Elasticsearch by OpenDistro

Installation for OpenDistro elasticsearch.

OpenDistro Elasticsearch Installation

Initial Configuration

Copy default fluentd and opendistro helm config files (and edit if desired)

```
cp ./seldon-deploy-install/prerequisites-setup/efk/fluentd-values.yaml fluentd-values.
↪yaml
cp ./seldon-deploy-install/prerequisites-setup/efk/values-opendistro.yaml values-
↪opendistro.yaml
```

(continues on next page)

(continued from previous page)

Ensure Required Namespaces Exist

We'll be installing in the `seldon-logs` namespace. We'll also set up some config in the `seldon-system` namespace.

```
kubectl create namespace seldon-logs || echo "namespace seldon-logs exists"
kubectl create namespace seldon-system || echo "namespace seldon-system exists"
```

Create Secrets

Auth is required for OpenDistro. That means components interacting with it will need secrets, including Seldon Deploy.

We'll need secrets in the `seldon-logs` namespace (for a request logger) and `seldon-system` for deploy.

Using the defaults we can set:

```
ELASTIC_USER=admin
ELASTIC_PASSWORD=admin

kubectl create secret generic elastic-credentials -n seldon-logs \
  --from-literal=username="${ELASTIC_USER}" \
  --from-literal=password="${ELASTIC_PASSWORD}" \
  --dry-run=client -o yaml | kubectl apply -f -

kubectl create secret generic elastic-credentials -n seldon-system \
  --from-literal=username="${ELASTIC_USER}" \
  --from-literal=password="${ELASTIC_PASSWORD}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

Using alternative creds is covered in the docs for opendistro and can be set on kibana and in the fluentd values file.

Elasticsearch

Install Elasticsearch and Kibana using the following script

```
git clone https://github.com/opendistro-for-elasticsearch/opendistro-build
cd opendistro-build/helm/opendistro-es/
git fetch --all --tags
git checkout tags/v1.13.2
helm package .
helm upgrade --install elasticsearch opendistro-es-1.13.2.tgz --namespace=seldon-logs␣
↪--values=../../../values-opendistro.yaml
```

You can wait for it to come up with

```
kubectl rollout status -n seldon-logs deployment/elasticsearch-opendistro-es-kibana
```

Fluentd

Install fluentd with this:

```
helm upgrade --install fluentd fluentd-elasticsearch --version 9.6.2 --
↪namespace=seldon-logs --values=../../../fluentd-values.yaml --repo https://kiwigrid.
↪github.io
```

Note that if alternative creds are used then these need to be set in the fluentd helm values file.

Configure Seldon Deploy

Following `helm` values needs to be set in `deploy-values.yaml` when using Elastisearch by OpenDistro:

```
requestLogger:
  create: true
  elasticsearch:
    host: elasticsearch-opendistro-es-client-service.seldon-logs.svc.cluster.local
    port: "9200"
    protocol: https

elasticsearch:
  basicAuth: true
  url: https://elasticsearch-opendistro-es-client-service.seldon-logs.svc.cluster.
↪local:9200
```

Configure EFK Ingress (Optional)

Kibana

It can be useful to access kibana's UI without having to port-forward.

To do this create the following `VirtualService` for `Kibana` to enable its ingress

```
cat << EOF > kibana-vs.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: kibana
  namespace: seldon-logs
spec:
  gateways:
    - istio-system/seldon-gateway
  hosts:
    - '*'
  http:
    - match:
        - uri:
            prefix: /kibana/
      rewrite:
        uri: /
      route:
        - destination:
            host: elasticsearch-opendistro-es-kibana-svc
```

(continues on next page)

(continued from previous page)

```
            port:
              number: 443
EOF

kubectl apply -f kibana-vs.yaml
```

Verify Install

You can go to /kibana/ from the istio ingress endpoint (the external ip/host of istio-ingressgateway in istio-system namespace).

Or you can verify elastic direct. First port-forward:

```
kubectl port-forward -n seldon-logs svc/elasticsearch-opendistro-es-client-service␣
↪9200
```

Then verify with below, using your user and pass (a browser can also be used).

```
curl -XGET -k https://localhost:9200 -u admin:admin
```

Elasticsearch by Elastic

Installation for elastic-provided elasticsearch.

Elasticsearch Installation

Initial Configuration

Copy default fluentd-config (and edit as required)

```
cp ./seldon-deploy-install/prerequisites-setup/efk/fluentd-values.yaml fluentd-values.
↪yaml
```

Elasticsearch

Install Elasticsearch using the following script

```
kubectl create namespace seldon-logs || echo "namespace seldon-logs exists"

helm upgrade --install elasticsearch elasticsearch \
    --version 7.6.0 \
    --namespace seldon-logs \
    --set service.type=ClusterIP \
    --set antiAffinity="soft" \
    --repo https://helm.elastic.co \
    --set image=docker.elastic.co/elasticsearch/elasticsearch-oss

kubectl rollout status statefulset/elasticsearch-master -n seldon-logs
```

Fluentd and Kibana

If not using auth (not by default) then set `elasticsearch.auth.enabled` to false in the fluentd values file.

Install fluentd and kibana using the following script

```
helm upgrade --install fluentd fluentd-elasticsearch \
    --version 8.0.0 \
    --namespace seldon-logs -f fluentd-values.yaml \
    --repo https://kiwigrid.github.io

helm upgrade --install kibana kibana \
    --version 7.6.0 \
    --namespace seldon-logs \
    --set service.type=ClusterIP \
    --repo https://helm.elastic.co \
    --set image=docker.elastic.co/kibana/kibana-oss

kubectl rollout status deployment/kibana-kibana -n seldon-logs
```

Configure Seldon Deploy

Following `helm` values needs to be set in `deploy-values.yaml` when using Elastisearch by Elastic:

```
requestLogger:
  create: true
  elasticsearch:
    host: elasticsearch-master.seldon-logs.svc.cluster.local
    port: "9200"
    protocol: http

elasticsearch:
  basicAuth: false
  url: http://elasticsearch-master.seldon-logs.svc.cluster.local:9200
```

Auth is not required for the elastic-provided version of elasticsearch. The `elasticsearch.basicAuth` option in the Seldon Deploy helm chart can be set to `false`, as illustrated above.

Authentication

If authentication is required to access your ElasticSearch cluster, you will need to configure your credentials so that Seldon can access it. To do this, you can provide your ElasticSearch user and password through a secret. By default, Seldon will look for a secret named `elastic-credentials`.

As an example, if we assume that ElasticSearch can be accessed using the `admin` / `admin` credentials, we could create the relevant secrets as:

```
ELASTIC_USER=admin
ELASTIC_PASSWORD=admin

kubectl create secret generic elastic-credentials -n seldon-logs \
  --from-literal=username="${ELASTIC_USER}" \
  --from-literal=password="${ELASTIC_PASSWORD}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

(continues on next page)

(continued from previous page)

```
kubectl create secret generic elastic-credentials -n seldon-system \
  --from-literal=username="${ELASTIC_USER}" \
  --from-literal=password="${ELASTIC_PASSWORD}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

Configure EFK Ingress (Optional)

Kibana

It can be useful to access kibana's UI without having to port-forward.

To expose kibana externally it needs to have its own path. This means a custom values file:

```
extraEnvs:
  - name: SERVER_BASEPATH
    value: "/kibana"
```

That should be referenced with `-f` as an additional parameter on the previous helm install command.

Then create a following `VirtualService` for `Kibana` to enable its ingress

```
cat << EOF > kibana-vs.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: kibana
  namespace: seldon-logs
spec:
  gateways:
  - istio-system/seldon-gateway
  hosts:
  - '*'
  http:
  - match:
    - uri:
        prefix: /kibana/
    rewrite:
      uri: /
    route:
    - destination:
        host: kibana-kibana
        port:
          number: 5601
EOF

Kubectl apply -f kibana-vs.yaml
```

GitOps

GitOps setup with ArgoCD

This page provides a detailed overview of how to set up the GitOps functionality for Seldon Deploy. Setting up GitOps is highly recommended.

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components are installed.

Seldon Deploy leverages ArgoCD to implement GitOps in your cluster. You can read more about Seldon Deploy's GitOps integration in the *architecture* and *operation* sections.

This documentation page will walk you through the different steps required to set up ArgoCD and GitOps in your cluster. Note that this guide assumes a fresh cluster without ArgoCD, thus it must only be treated as a general reference. Feel free to adapt these steps to your particular infrastructure.

ArgoCD Installation

Since this guide assumes that you currently don't have ArgoCD installed in your cluster, the first step will be on installing and configuring it. You can treat these steps as a quick way to get started with ArgoCD. Therefore, it is **highly encouraged** to check ArgoCD's official documentation for further configuration.

To install and configure ArgoCD we will first install their official K8s manifest into our cluster. For version `v1.6.2` of ArgoCD, this can be done as follows:

```
ARGOCD_VERSION=v1.6.2

kubectl create namespace argocd || echo "namespace exist"
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/$
↪{ARGOCD_VERSION}/manifests/install.yaml

kubectl rollout status deployment/argocd-application-controller -n argocd
kubectl rollout status deployment/argocd-repo-server -n argocd
kubectl rollout status deployment/argocd-server -n argocd
kubectl rollout status deployment/argocd-redis -n argocd
kubectl rollout status deployment/argocd-dex-server -n argocd
```

ArgoCD Configuration

Now that we have ArgoCD installed in our cluster, the next step will involve configuring it so that:

- The admin user credentials get changed.
- SSL termination is disabled in ArgoCD.
- ArgoCD gets exposed under the `/argocd/` path of our load balancer.

Note that these steps must be treated just as a guideline. Therefore, you may need to adapt them to your infrastructure. For more information on ArgoCD's configuration, we suggest checking ArgoCD's official documentation.

Admin user and password

Upon the initial installation of ArgoCD, it is recommended to change its user and password. This guideline will show how to change ArgoCD's default admin user as an example. However, it's also possible to hook ArgoCD to a centralised OIDC provider. This could be the same as *Seldon Deploy's OIDC provider*, providing a Single Sign-On across your entire stack.

The following command will assume that we want to set the admin credentials as `admin` // `$ARGOCDINITIALPASS`. With this in mind, we can set those credentials as:

```
ARGOCDINITIALPASS=12341234
kubectl patch secret \
    -n argocd argocd-secret \
    -p '{"stringData": { "admin.password": "'$(htpasswd -bnBC 10 "" $
↪{ARGOCDINITIALPASS} | tr -d ':\n')'"}}'
```

This command requires `htpasswd` that can be installed on Ubuntu, Debian and Mint systemms with:

```
sudo apt install apache2-utils
```

SSL Termination

As a general choice, we recommend to handle SSL termination at the ingress / load balancer level. This simplifies the components setup within the cluster.

Following this approach, we will need to disable the SSL termination in ArgoCD. Otherwise, ArgoCD will expect to receive SSL traffic by default. To do this, we can ask ArgoCD to run in `insecure` mode by running the command below:

```
kubectl patch deploy argocd-server \
    -n argocd \
    -p '[{"op": "add", "path": "/spec/template/spec/containers/0/command/-", "value":
↪"--insecure"}]' \
    --type json
```

Exposing ArgoCD through our ingress

There are different ways to access ArgoCD. However, the most convenient is usually to expose it through our ingress layer so that it can be accessed as a different subpath (e.g. `/argocd/`). This will make the ArgoCD API and UI reachable as `http(s)://<load-balancer-domain>/argocd/`.

This will need to be configured in a couple of places.

1. On one hand, we need to tell ArgoCD to expect its static assets to be served under the `/argocd/` path. This can be done by running the following command:

   ```
   kubectl patch deploy argocd-server \
       -n argocd \
       -p '[{"op": "add", "path": "/spec/template/spec/containers/0/command/-",
   ↪"value": "--rootpath"}, {"op": "add", "path": "/spec/template/spec/containers/0/
   ↪command/-", "value": "/argocd"}]' \
       --type json
   ```

2. On the other hand, we will need to create a `VirtualService` that maps the `/argocd/` path to ArgoCD's service. Note that this assumes that we are using Istio as an ingress layer. To expose ArgoCD, it should be enough to apply the following `VirtualService` resource to our cluster:

```
cat << EOF > ./argocd-vs.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: argocd-server
  namespace: argocd
spec:
  gateways:
  - istio-system/seldon-gateway
  hosts:
  - '*'
  http:
  - match:
    - uri:
        prefix: /argocd/
    route:
    - destination:
        host: argocd-server
        port:
          number: 80
EOF
```

If we assume that the above has been saved as a file called `argocd-vs.yaml`, we can then apply it as:

```
kubectl apply -f argocd-vs.yaml
```

Accessing ArgoCD

After configuring ArgoCD, we should now be able to reach ArgoCD under:

```
http(s)://<load-balancer-domain>/argocd/
```

In a standard setup, you should be able to obtain your load balancer domain by checking the Istio resources created in your cluster. You can do this as:

```
ISTIO_INGRESS=$(\
  kubectl get svc -n istio-system istio-ingressgateway \
    -o jsonpath='{.status.loadBalancer.ingress[0].ip}'\
)
ISTIO_INGRESS+=$(\
  kubectl get svc -n istio-system istio-ingressgateway \
  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'\
)

echo $ISTIO_INGRESS
```

Alternatively, if your ingress layer doesn't expose an external IP address or domain, it's also possible to *port-forward* a local port into ArgoCD. For example, to map your local `localhost:8080` (or `localhost:8080/argocd/` if you defined the `rootpath` above) domain to ArgoCD, you can run:

```
kubectl port-forward -n argocd svc/argocd-server 8080:80
```

Once we know our load balancer domain, we can then access ArgoCD through its UI or API. For the latter, we will leverage ArgoCD's CLI, called `argocd`. Below, you can find instructions on how to set up and access both of them.

CLI

UI

ArgoCD ships with a CLI tool called `argocd`. This tool allows you to interact with ArgoCD from the command line. The rest of this guide will provide command examples that can be used to configure ArgoCD using their `argocd` tool. However, it's also possible to perform the same actions from the ArgoCD UI.

To use the `argocd` CLI, we will need to:

1. Install the `argocd` CLI locally. This can be achieved by running the following commands:

```
ARGOCD_VERSION=v1.6.2
wget -q -O argocd "https://github.com/argoproj/argo-cd/releases/download/${ARGOCD_
↪VERSION}/argocd-linux-amd64"
chmod +x ./argocd
```

2. Authenticate the CLI against our ArgoCD instance. Assuming that our load balancer domain can be found under `$ISTIO_INGRESS`, it should be enough to run the command below.

```
./argocd login \
  "$ISTIO_INGRESS:80" \
  --username admin \
  --password ${ARGOCDINITIALPASS} \
  --insecure \
  --grpc-web-root-path /argocd
```

ArgoCD exposes a UI that can be used to configure your GitOps repository. Assuming that you have followed the *ArgoCD setup guidelines above*, this UI should be accessible under the `/argocd/` path of your load balancer domain.

After accessing the UI, you should be prompted with a login form asking for the admin credentials. You should be able to access this UI by using ArgoCD's user / password combination (`admin` and `$ARGOCDINITIALPASS` by default, respectively).

You can learn more about how to navigate ArgoCD's UI in their documentation.

GitOps for logging components

The request logger is an infrastructure component that runs in the *centralised logging namespace* (named as `seldon-logs` by default). Alongside this, Triggers can also be created for detectors and these tied to **particular models**. Thus is good to back them through a GitOps environment.

With this goal in mind, it's encouraged to **create a system-level GitOps environment** to track these infrastructure resources. To do this, you can follow the *instructions to create a new GitOps environment*, using the `seldon-logs` namespace as target. Note that if you've configured Seldon Deploy to use a different centralised namespace for logging, you'll need to point to that namespace instead.

Once the logging GitOps environment is created, Seldon Deploy should be able to pick it up and use it without any further configuration.

GitOps environment creation

Now that ArgoCD has been *installed and configured*, we can then proceed to create a new GitOps environment. This environment will provide a Kubernetes namespace to deploy our machine learning models, which is backed by a GitOps repository.

This section will walk you through how to create a new GitOps-tracked environment for a namespace with name `$namespace`. The GitOps repository which will act as the source of truth for this environment will exist under `$GIT_REPO_URL`. Note that `$GIT_REPO_URL` should not contain the `http` or `https` protocol schema.

Git Repository

The only requirements for the Git repository are:

- It contains a non-empty folder **named as the namespace** (i.e. `./$namespace`).
- It is accessible from within the cluster. This can be verified using this one-liner command:

```
kubectl run --quiet=true -it --rm git-clone --image=radial/busyboxplus:git  --
↪restart=Never -- git clone https://$GIT_USER:$GIT_TOKEN@$GIT_REPO_URL
```

ArgoCD Resources

In order for ArgoCD to keep track of our new environment, it's necessary to configure a *Project* and an *Application*. These are concepts native to ArgoCD. We suggest checking the ArgoCD official documentation for more information about them.

ArgoCD projects and applications can be created via the ArgoCD UI, `argocd` CLI tool and through CRDs. As declarative approach guarantees best reproducibility this is the approach that we recommend.

We create ArgoCD project by creating the following `AppProject` resources named `argocd-project.yaml`

```
cat << EOF > ./argocd-project.yaml
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: seldon
  namespace: argocd
spec:
  description: Seldon Deploy Project
  sourceRepos:
  - https://${GIT_REPO_URL}
  destinations:
  - namespace: ${namespace}
    server: https://kubernetes.default.svc
  clusterResourceWhitelist:
  - group: '*'
    kind: '*'
  roles:
  - name: seldon-admin
    policies:
    - p, proj:seldon:seldon-admin, applications, get, seldon/*, allow
    - p, proj:seldon:seldon-admin, applications, create, seldon/*, allow
    - p, proj:seldon:seldon-admin, applications, update, seldon/*, allow
    - p, proj:seldon:seldon-admin, applications, delete, seldon/*, allow
```

(continues on next page)

(continued from previous page)

```
    - p, proj:seldon:seldon-admin, applications, sync, seldon/*, allow
EOF

kubectl apply -f argocd-project.yaml
```

Similarly we create the ArgoCD `Application` resource:

```
ARGO_APP_NAME=seldon-gitops-"${namespace}"

cat << EOF > ./argocd-application.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: ${ARGO_APP_NAME}
  namespace: argocd
spec:
  project: seldon
  destination:
    namespace: ${namespace}
    server: https://kubernetes.default.svc
  source:
    directory:
      recurse: true
    path: ${namespace}
    repoURL: https://${GIT_REPO_URL}
  syncPolicy:
    automated: {}
EOF

kubectl apply -f argocd-application.yaml
```

Kubernetes Model Namespace

To use a namespace with GitOps, you will need to annotate it with the Git repository URL (using key `git-repo`) and enable it for GitOps access by adding the `git-repo` label. As an example, to enable a namespace named `$namespace`, you could run the following commands:

```
kubectl create ns $namespace || echo "namespace $namespace already exists"

kubectl label ns $namespace seldon.restricted=false --overwrite=true
kubectl label namespace $namespace seldon.gitops=enabled --overwrite=true
kubectl annotate namespace $namespace git-repo="https://${GIT_REPO_URL}" --
↪overwrite=true
```

Note: If your ArgoCD application does not follow `seldon-gitops-${namespace}` naming convention you can label the namespace accordingly

```
kubectl label namespace $namespace argocdapp=${ARGO_APP_NAME} \--overwrite=true
```

If you intend to use Batch jobs on the namespace then you'll need additional config. You can visit the *Argo section* of the docs for more details.

Configuring Git Credentials

The last required step is to provide the relevant Git credentials to access our GitOps repository. Note that these credentials need to be set at both the ArgoCD and Seldon Deploy level.

ArgoCD

You can configure your GitOps repository access in ArgoCD using either SSH or user / password credentials. These credentials can be provided using the `argocd` CLI tool, or directly through the ArgoCD UI.

SSH (CLI)

User / Password (CLI)

UI

Assuming that the *`argocd` CLI tool is already authenticated*, and that `$GIT_SSH_PATH` and `$GIT_TOKEN` represent our GitOps repository user and password, we should be able to run:

```
./argocd repo add "ssh://${GIT_REPO_URL}" \
    --ssh-private-key-path ${GIT_SSH_PATH} \
    --upsert
```

Assuming that the *`argocd` CLI tool has already been authenticated*, and that `${GIT_USER}` and `${GIT_TOKEN}` represent our GitOps repository user and password, we should be able to run:

```
./argocd repo add "https://${GIT_REPO_URL}" \
    --username ${GIT_USER} \
    --password ${GIT_TOKEN} \
    --upsert
```

ArgoCD exposes a UI that can be used to configure your GitOps repository. Assuming that you have followed the *ArgoCD setup guidelines above*, this UI should be accessible under the `/argocd` path. You should be able to access this UI by using ArgoCD's user / password combination (`admin` and `12341234` by default, respectively).

You can learn more about how to configure repositories using ArgoCD's UI in their documentation.

Seldon Deploy

Seldon Deploy also requires access to the repository. This access will be used to monitor the status of the current resources in the cluster, as well as to create new ones.

To configure our Git credentials in Seldon Deploy, we will follow these steps:

1. Create a Kubernetes secret containing our credentials, either as a SSH key or a User / Password combination. This secret can have any arbitrary name, but must live in the same namespace as Seldon Deploy.

   SSH

   User / Password

   If we assume that our private key is present under `$GIT_SSH_PATH`, we can create the credentials secret as:

   ```
   kubectl create secret generic git-creds -n seldon-system \
     --from-file ${GIT_SSH_PATH} \
     --from-file ${GIT_KNOWN_HOSTS_PATH} \
     --from-literal=passphrase="${GIT_SSHKEY_PASSPHRASE}" \
   ```

(continues on next page)

(continued from previous page)

```
  --from-literal=username="${GIT_USER}" \
  --from-literal=email="${GIT_EMAIL}"  \
  --dry-run=client -o yaml | kubectl apply -f -
```

Note that the `passphrase` field can be left empty if they SSH key doesn't have a passphrase.

We can create the credentials secret using a User / Password combination (or User / Personal Access Token) as:

```
kubectl create secret generic git-creds -n seldon-system \
  --from-literal=username="${GIT_USER}" \
  --from-literal=token="${GIT_TOKEN}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

2. Update Seldon Deploy's configuration to point to our newly created secret. In particular, we will need to modify the `gitops` section of the values of the Seldon Deploy Helm chart. Here, we will need to set the `gitops.argocd.enabled` flag to `true`, and we will point the `gitops.git.secret` field to the right secret name (e.g. `git-creds` if we've followed the commands above).

```
gitops:
  git:
    secret: git-creds
  argocd:
    enabled: true
```

(Optional) Webhooks Configuration

By default, the resource synchronisation against the GitOps repository will happen on a poll basis. That is, Seldon Deploy will check the repo periodically for updates.

The main caveat of this approach is that there may be a small delay between taking an action and seeing it reflected in the cluster / UI. To work around this, you can configure a set of webhooks in the cluster in the that will get pinged everytime there is a change in the cluster. Note that this is an **optional step**, and its feasibility may depend on your networking infrastructure and your Git provider.

Hint: If webhooks are not arriving at its destination it may be helpful to enable webhook's history.

It may be necessary to also select `Skip certificate verification` option.

You can find instructions below on how to set up these webhooks for common Git providers

GitHub

Bitbucket

First we need to configure the webhook that will be sending updates to the Seldon Deploy endpoint:

```
whIp=$(kubectl get service seldon-deploy-webhook -o jsonpath='{.status.loadBalancer.
↪ingress[*].ip}')
curl -u ${GIT_USER}:${GIT_TOKEN} \
   -v -H "Content-Type: application/json" \
   -d '
    {
      "name": "web",
      "active": true,
```

(continues on next page)

(continued from previous page)

```
      "events": ["*"],
      "config": {
        "url": "http://${whIp}/api/git-webhook",
        "content_type": "json"
      }
    }' \
   https://${GIT_API}/repos/${GIT_USER}/${REPONAME}/hooks
```

Then we set up the webhook that will be sending the updates to ArgoCD.

```
curl -u ${GIT_USER}:${GIT_TOKEN} \
   -v -H "Content-Type: application/json" \
   -d '\
    {
      "name": "web",
      "active": true,
      "events": ["*"],
      "config": {
        "url": "https://${ARGOCDURL}/api/webhook",
        "content_type": "json",
        "secret": "${ARGOCDINITIALPASS}",
        "insecure_ssl": "1"
      }
    }' \
   https://${GIT_API}/repos/${GIT_USER}/${REPONAME}/hooks
```

First we need to configure the webhook that will be sending updates to the Seldon Deploy endpoint:

```
whIp=$(kubectl get service seldon-deploy-webhook -o jsonpath='{.status.loadBalancer.
↪ingress[*].ip}')
curl -u ${GIT_USER}:${GIT_TOKEN} \
   -v -H "Content-Type: application/json" \
   -d '
    {
        "description": "web",
        "events": ["repo:push"],
        "url": "'"http://${whIp}/api/git-webhook"'",
        "active": true
    }' \
  https://${GIT_API}/2.0/repositories/${GIT_USER}/${REPONAME}/hooks
```

Then we set up the webhook that will be sending the updates to ArgoCD.

```
curl -u ${GIT_USER}:${GIT_TOKEN} \
   -v -H "Content-Type: application/json" \
   -d '
    {
        "description": "web",
        "events": ["repo:push"],
        "url": "'"https://${ARGOCDURL}/api/webhook"'",
        "active": true
    }' \
  https://${GIT_API}/2.0/repositories/${GIT_USER}/${REPONAME}/hooks
```

Troubleshooting

You should see one application per gitops namespace when running the below:

```
kubectl get application -n argocd
```

You can check the status of an argocd app with `kubectl get application -n argocd seldon-gitops-<namespace_name> -o yaml`.

The argocd UI (described above) can also be used to inspect applications. You can trigger a manual sync from there. If a particular application is missing or in error try running its setup again in order to debug.

Example Script for Namespace Setup

Download installation resources as explained *here*

Copy the sd-setup directory containing `gitops-setup.sh`

```
cp -r ./seldon-deploy-install/sd-setup/ sd-setup/
```

The gitops-setup.sh is an example script for setting up gitops in the cluster by installing argocd and configuring namespaces. It can be used for reference.

See the *trial installation* docs for more on how to run that script. More context is provided in *Namespace Setup*

```

```

Authentication

Configure App Level Authentication for Seldon Deploy

App Level Authentication

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components are installed.

Seldon Deploy can be setup with an app level authentication with an `OIDC` provider using the auth code flow. This would be particularly useful when a gateway level authentication is not setup for the kubernetes cluster.

This feature can be activated by configuring the values file of the Seldon Deploy helm chart. This is done by setting the variable `enableAppAuth` boolean to "true" and further providing the OIDC configurations as env variables to connect to `OIDC` providers like `Keycloak` or `Dex`.

```
# boolean to enable app-level auth (defaults to "false")
enableAppAuth: true
```

Add config / env variables

Before we run deploy install using the helm chart, we need to make sure that add the OIDC configuration to the env section:

```
env:
  OIDC_PROVIDER: ... # oidc providerURL
  CLIENT_ID: ... # oidc client ID
  CLIENT_SECRET: ... # oidc client secret
  REDIRECT_URL: ... # `${oidc_redirect_url}/seldon-deploy/auth/callback`
  OIDC_SCOPES: ... # oidc scopes (defaults to "profile email groups")
  USERID_CLAIM_KEY: ... # claim to be used as userid (defaults to "preferred_username
↪")
```

Hint: Do not forget to run `helm upgrade seldon-deploy ...` as described on Seldon Deploy configuration section.

Keycloak reference installation

An OIDC provider is needed to authenticate to. A reference installation of keycloak is included with scripts under the following directory

```
seldon-deploy-install/prerequisites-setup/keycloak
```

However, customisation of created users, passwords and tokens is **highly** recommended. Following reverence installation this should be the configuration values:

```
env:
  CLIENT_ID: "deploy-server"
  CLIENT_SECRET: "deploy-secret"
  OIDC_PROVIDER: "http://${YOUR_INGRESS}/auth/realms/deploy-realm"
  REDIRECT_URL: "http://${YOUR_INGRESS}/seldon-deploy/auth/callback"
```

With Istio ingress may be obtained with

```
ISTIO_INGRESS=$(kubectl get svc -n istio-system istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}')
ISTIO_INGRESS+=$(kubectl get svc -n istio-system istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].hostname}')

echo "OIDC_PROVIDER: http://${ISTIO_INGRESS}/auth/realms/deploy-realm"
echo "REDIRECT_URL: http://${ISTIO_INGRESS}/seldon-deploy/auth/callback"
```

Groups/LDAP Configuration

Some customers choose to get groups from LDAP. See *LDAP section* on that.

Identity Brokering

Sometimes there's an existing identity server that doesn't support OIDC and instead supports SAML.

Seldon does not support SAML directly but keycloak can be used to broker to a SAML server.

Debugging

There's some details on debugging the token content in the *LDAP section*.

Often the issue is the auth configuration rather than Seldon specifically. In those cases it can be useful to connect a different application that dumps more details.

We provide an example app for debugging purposes. It can be run on the cluster alongside Seldon Deploy.

Or you can edit your Seldon Deploy installation and replace the docker image with that one. To do this, first find the deployment for Seldon Deploy and find what image it is using:

```
kubectl get deployment -n seldon-system seldon-deploy -o jsonpath="{..image}"
```

Then make a note of that and save the note. Next edit the deployment with `kubectl edt deployment -n seldon-system seldon-deploy`. Change the image to `seldonio/oauth-test-tool:0.1`.

Now when you login to Seldon Deploy it will actually login to the test tool. Its pod logs will give more detail about the login process. After debugging and resolution then the image can be edited back to what it was before.

Minio

This page provides steps for installing minio.

Installation of Minio

We suggest to install with the minio helm chart.

```
MINIOUSER=minioadmin
MINIOPASSWORD=minioadmin

kubectl create ns minio-system
helm repo add minio https://helm.min.io/
helm upgrade --install minio minio/minio \
    --set accessKey=${MINIOUSER} \
    --set secretKey=${MINIOPASSWORD} \
    --namespace minio-system
```

Finally, we create VirtualService named `minio-vs.yaml`

```
cat << EOF > ./minio-vs.yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: minio
  namespace: minio-system
spec:
  gateways:
    - istio-system/seldon-gateway
  hosts:
    - '*'
  http:
    - match:
        - uri:
            prefix: /minio/
      route:
        - destination:
            host: minio
            port:
              number: 9000
EOF
```

and apply it with

```
kubectl apply -f minio-vs.yaml
```

Per Namespace Setup

See the *Argo section* for additional per-namespace setup for batch jobs. The secret suggested there can also be used for models.

Verify Install

You can go to /minio/ from the istio ingress endpoint (the external ip/host of istio-ingressgateway in istio-system namespace) and login with the credentials used above.

Or you can port-forward:

```
kubectl port-forward -n minio-system svc/minio 9000:9000
```

Then go to `localhost:9000` in the browser.

Argo

This page provides steps for installing argo.

Installation of Argo

We suggest to install argo in line with the argo instructions. At the time of writing these are:

```
kubectl create namespace argo
kubectl apply -n argo -f https://raw.githubusercontent.com/argoproj/argo-workflows/
↪stable/manifests/install.yaml
```

Per Namespace Setup

If you intend to use Batch jobs on the namespace then you'll need to create a service account for this:

```
namespace=seldon

kubectl apply -n ${namespace} -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: workflow
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - "apps"
  resources:
  - deployments
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - pods/log
  verbs:
  - "*"
- apiGroups:
  - machinelearning.seldon.io
  resources:
  - "*"
  verbs:
  - "*"
EOF

kubectl create -n ${namespace} serviceaccount workflow

kubectl create rolebinding -n ${namespace} workflow --role=workflow --serviceaccount=$
↪{namespace}:workflow
```

And also configure a secret for batch jobs to communicate with S3 or minio (here assumed to be in minio-system namespace). Batch job processor in Seldon Deploy uses Storage Initializer mechanism similar to one used on the Pre Packaged model servers.

Rclone based storage initializer (default)

Leaving the default helm value for `batchjobs.storageInitailizer`:

```
batchjobs:
  storageInitializer:
    image: seldonio/rclone-storage-initializer:1.9.1
```

will result in rclone-based storage initializer being used. Rclone offers a compatibility with over 40 different cloud storage products and therefore is the default choice.

For the *minio installation* secret is as follows:

```
MINIOUSER=minioadmin
MINIOPASSWORD=minioadmin
namespace=seldon

kubectl apply -n ${namespace} -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: seldon-job-secret
type: Opaque
stringData:
  RCLONE_CONFIG_S3_TYPE: s3
  RCLONE_CONFIG_S3_PROVIDER: minio
  RCLONE_CONFIG_S3_ENV_AUTH: "false"
  RCLONE_CONFIG_S3_ACCESS_KEY_ID: ${MINIOUSER}
  RCLONE_CONFIG_S3_SECRET_ACCESS_KEY: ${MINIOPASSWORD}
  RCLONE_CONFIG_S3_ENDPOINT: http://minio.minio-system.svc.cluster.local:9000
EOF
```

Using custom storage initializer

If for some reason you would like to use different storage initializer, e.g. kfserving storage initializer you can set this by modifying the mentioned `deploy-values.yaml`:

```
batchjobs:
  storageInitializer:
    image: gcr.io/kfserving/storage-initializer:v0.4.0
```

The corresponding secret would also need to be modified:

```
MINIOUSER=minioadmin
MINIOPASSWORD=minioadmin
namespace=seldon

kubectl apply -n ${namespace} -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: seldon-job-secret
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: ${MINIOUSER}
  AWS_SECRET_ACCESS_KEY: ${MINIOPASSWORD}
```

(continues on next page)

(continued from previous page)

```
  AWS_ENDPOINT_URL: http://minio.minio-system.svc.cluster.local:9000
  USE_SSL: "false"
EOF
```

Running on GKE or inside Kind cluster

Note, if running inside `kind` cluster or on GKE one must patch Argo's config

```
kubectl patch -n argo configmap workflow-controller-configmap --type merge \
    -p '{"data": {"config": "containerRuntimeExecutor: k8sapi"}}'
```

Verification and Debugging

You can check the argo install by going to the argo UI. First port-forward:

```
kubectl port-forward -n argo svc/argo-server 2746
```

Then go to `http://localhost:2746/` in the browser.

If argo is setup correctly then you should be able to run the *batch demo*.

To see running jobs, you can use the argo UI or its CLI, if you install that. You can list jobs in the namespace with `argo list -n <namespace>`. An `argo get` tells you the pod names of the steps.

To see logs for a running job, go to the relevant pod. If you don't have the argo CLI you can work out the pod name as there should be a pod in the namespace with a running status and a name similar to the model name.

PostgreSQL Persistence for Model Metadata

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components are installed.

We use PostgreSQL for persisting model metadata information.

Seldon Deploy Configuration

Enabling/disabling the PostgreSQL dependency in Seldon Deploy can be done with setting the following Helm variable - `metadata.pg.enabled`. If it is set to `false` Seldon Deploy will not attempt to connect to a PostgreSQL database, but all model metadata functionality will be unavailable. If `metadata.pg.enabled` is `true`, then Seldon Deploy will expect a `metadata-postgres` Kubernetes secret to be present in the namespace where Seldon Deploy is running. This secret needs to contain the information for connecting to a PostgreSQL database. The structure of the secret is:

```
kind: Secret
apiVersion: v1
data:
  dbname: the_name_of_the_database_to_use_for_model_metadata
```

(continues on next page)

(continued from previous page)

```
  host: the_database_host
  user: the_database_user_to_use_to_authenticate
  password: the_database_password_to_use_to_authenticate
  port: the_port_the_database_is_exposed_on
  sslmode: the_sslmode
```

Installation

PostgreSQL can be installed in many different ways - using managed solutions by cloud providers, or running it in Kubernetes.

Bringing your own PostgreSQL

One option is to use PostgreSQL outside of the Kubernetes cluster that runs Seldon Deploy. If you already have a database you want to use with Seldon Deploy running on prem or in the cloud you can add the connection information in the `metadata-postgres` secret in the namespace Seldon Deploy is running like this substituting the values with the ones of your database:

```
kubectl create secret generic -n seldon-system metadata-postgres \
--from-literal=user=your_user \
--from-literal=password=your_password \
--from-literal=host=your.postgres.host \
--from-literal=port=5432 \
--from-literal=dbname=metadata \
--from-literal=sslmode=require \
--dry-run=client -o yaml \
| kubectl apply -n seldon-system -f -
```

In the next sections we explore how you can start using a managed PostgreSQL in AWS and GCP and connect it with Seldon Deploy.

Amazon RDS

Amazon RDS provides a managed PostgreSQL solution that can be used for Seldon Deploy's Model Metadata Storage. For setting up RDS for the first time you can follow the docs here.

Some important point to remember while setting up RDS:

- Make sure the instance is accessible from Seldon Deploy. If Seldon Deploy is not on the same VPC, make sure the VPC used by RDS has a public subnet as discussed here.
- Make sure the security group used for accessing the RDS instances allow inbound and outbound traffic from and to Seldon Deploy. Setting up security groups for RDS is discussed here.

Once you have a running PostgreSQL instance, with a database and a user created you can configure Seldon Deploy by adding the `metadata-postgres` secret as discussed in *the previous section*.

To manage backups see the official documentation. Here is more documentation on other best practices around RDS.

Google SQL

GCP provides a managed PostgreSQL solution that can be used for Seldon Deploy's Model Metadata Storage. For setting up Google SQL for the first time you can follow the docs here.

For connection instructions follow the official documentation. Make sure that the instance is accessible from Seldon Deploy. If using the public IP generated for the instance make sure the network that runs Seldon Deploy is part of the Cloud SQL authorized networks by following this guide.

Once you have a running PostgreSQL instance, with a database and a user created you can configure Seldon Deploy by adding the `metadata-postgres` secret as discussed in *the previous section*.

Running PostgreSQL in Kubernetes

You can also run PostgreSQL in the Kubernetes cluster that runs Seldon Deploy. We recommend using the Zalando PostgreSQL operator to manage the PostgreSQL installation and maintenance. The official documentation can be seen here. Below we show an example deployment of a PostgreSQL cluster:

To install the Zalando operator you can run:

```
git clone https://github.com/zalando/postgres-operator.git
git checkout v1.6.3 # Use a tag to pin what we are using.
cd postgres-operator
kubectl create namespace postgres || echo "namespace postgres exists"
helm install postgres-operator ./charts/postgres-operator --namespace postgres
```

If you want to install the operator UI you can do it by following this doc.

To install a minimal PostgreSQL setup you can run:

```
cat <<EOF | kubectl apply -f -
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: seldon-metadata-storage
  namespace: postgres
spec:
  teamId: "seldon"
  volume:
    size: 5Gi
  numberOfInstances: 2
  users:
    seldon:  # database owner
    - superuser
    - createdb
  databases:
    metadata: seldon  # dbname: owner
  postgresql:
    version: "13"
EOF
```

For a more complex setup consisting of more users, databases, replicas, etc. please refer to the official documentation of the operator here.

Once the database instances have been created by the Zalando operator you can create the expected secret using the auto generated password:

```
kubectl get secret seldon.seldon-metadata-storage.credentials.postgresql.acid.zalan.
↪do -n postgres -o 'jsonpath={.data.password}' | base64 -d > db_pass
kubectl create secret generic -n seldon-system metadata-postgres \
  --from-literal=user=seldon \
  --from-file=password=./db_pass \
  --from-literal=host=seldon-metadata-storage.postgres.svc.cluster.local \
  --from-literal=port=5432 \
  --from-literal=dbname=metadata \
  --from-literal=sslmode=require \
  --dry-run=client -o yaml \
  | kubectl apply -n seldon-system -f -
rm db_pass
```

Configuring Seldon Deploy

Once you have your PostgreSQL database ready and the secret with credentials ready add to `deploy-values.yaml` following:

```
metadata:
  pg:
    enabled: true
    secret: metadata-postgres
```

Production operations on self-managed PostgreSQL

One of drawbacks of using self-hosted PostgreSQL rather than a managed solution is that you will need to handle operating the PostgreSQL cluster. Here is a list of some resources for best practices and how to handle some operations:

- Monitoring - deploying postgres exporter and hooking it up with your Prometheus monitoring solution is a common way of getting continuous monitoring of the instances.
- Backups - the Zalando operator provides setup of periodic backups in s3 compatible storage - https://postgres-operator.readthedocs.io/en/latest/administrator/#wal-archiving-and-physical-basebackups It also documents restoring state from backups - https://postgres-operator.readthedocs.io/en/latest/administrator/#restoring-physical-backups. **We strongly recommend setting backups if self-hosting PostgreSQL.**
- Version update - Zalando support cloning and in-place version updates - https://postgres-operator.readthedocs.io/en/latest/administrator/#minor-and-major-version-upgrade
- Increase storage size - https://postgres-operator.readthedocs.io/en/latest/user/#increase-volume-size

Authorization

Configure Open Policy Agent Authorization for Seldon Deploy

Authorization is an optional part of the platform which can be enabled or disabled based on your requirements.

Using namespace labels

Namespace labels can be used to authorize access to namespaced resources. For more details see the *docs*.

Setting Up Open Policy Agent authorization in Deploy

OPA authorization is experimental. To enable OPA authorization in Deploy the following values must be set in the Deploy Helm chart:

```
rbac:
  opa:
    enabled: true
    configMap: seldon-deploy-policies # default to this, change only if you want to␣
↪use another config map.
    projectAuthEnabled: true # enable if you want to authorize project based␣
↪resources (models)
```

This will tell Deploy to load the OPA policies stored in the given `rbac.opa.configMap` and use them to authorize requests.

Some important considerations when enabling OPA authorization:

- If the specified config map does not exist, Deploy will not start.
- If the config map does not contain policies, authorization requests will default to denying access.

For a detailed explanation of the schema of the policies, how to set up the config map, and how to migrate from namespace labels authorization to OPA authorization follow *the policy management guide*.

Note: `rbac.opa.projectAuthEnabled` requires the Model Catalog to be enabled as well since it authorizes resources in there, or resources containing models (deployments). To enable it follow *the postgres setup guide*.

KFserving Installation

This section will walk you through installation procedure of KFserving such that it is ready to be used with Seldon Deploy.

Important: Before starting the installation procedure, please download installation resources as explained *here* and make sure that all pre-requisites are satisfied.

This page also assumes that main *Seldon* components and *Istio* are installed.

Install KNative Serving

Knative Serving is required for KFServing. See *Knative section* for steps.

Install KFServing

Clone `KFserving` repository and checkout the right branch

```
KFSERVING_VERSION=v0.4.1
KFSERVING_RESOURCE_DIR=kfserving-${KFSERVING_VERSION}

git -C ${KFSERVING_RESOURCE_DIR} fetch origin || git clone https://github.com/
↪kubeflow/kfserving.git ${KFSERVING_RESOURCE_DIR}
#have to checkout commit rather than tag as v0.4.1 yaml is missing in v0.4.1 tag
git -C ${KFSERVING_RESOURCE_DIR} checkout f253dd529ba25c080ca458d8dc36fd0a19bbd473
```

Install and configure `KFserving`

```
cd ${KFSERVING_RESOURCE_DIR}
kubectl apply --validate=false -f ./install/${KFSERVING_VERSION}/kfserving.yaml ||␣
↪true
sleep 3
kubectl patch cm -n kfserving-system inferenceservice-config -p '{"data":{"explainers
↪":"{\n    \"alibi\":{\n        \"image\" : \"docker.io/seldonio/kfserving-alibi\",\
↪n        \"defaultImageVersion\" : \"0.4.1\"\n    }\n}","ingress":"{\n    \
↪"ingressGateway\" : \"istio-system/seldon-gateway\",\n    \"ingressService\" : \
↪"istio-ingressgateway.istio-system.svc.cluster.local\"\n}","logger":"{\n    \"image\
↪" : \"gcr.io/kfserving/logger:v0.4.0\",\n    \"memoryRequest\": \"100Mi\",\n    \
↪"memoryLimit\": \"1Gi\",\n    \"cpuRequest\": \"100m\",\n    \"cpuLimit\": \"1\",\n␣
↪   \"defaultUrl\": \"http://broker-ingress.knative-eventing.svc.cluster.local/
↪seldon-logs/default\"\n}"}}'
# Create self-signed certs because we dont have cert-manager
./hack/self-signed-ca.sh
```

Configure Seldon Deploy

Enable KFserving in Seldon Deploy by adding a following section to your `deploy-values.yaml` files

```
kfserving:
  protocol: "http"
  enabled: true
```

Hint: Note that if your istio Gateway uses https then set protocol to `https`.

3.7.5 Multi-tenant Installation

Guidance for multi-tenant clusters

3.7.5.1 Multi-tenant Guidance

We suggest running Seldon Deploy in cluster-wide mode. If the full stack of suggested components is used then this makes it simpler to scale by adding new namespaces dynamically for new departments/users.

If you are able to use cluster-wide permissions, you don't need to read this guide. Use one of the other install types.

The below is guidance on restricting the use of cluster-wide permissions.

Optional Components

Istio

Istio currently has very limited support for multi-tenancy.

There are alternative flavours of istio. OpenShift Service Mesh does have some multi-tenancy support.

An alternative is Ambassador, which can be installed namespaced.

If a non-default ingress setup is used then the `requestForm` and `curlForm` sections in the values file need to be configured with templates applicable to the ingress URL rules being used.

Argocd

From our testing, argocd does require read permissions at a cluster-wide level on all objects.

Write permissions can however be entirely namespaced. Using this means configuring roles for each new namespace, as explained in the Argocd GitHub issue.

Argocd is only required in order to use Gitops. Seldon Deploy can be used without Gitops.

We have a `MULTITENANT` flag in the deploy config file for a `Trial Installation`. This is used in the scripts provided in the *download of installation resources.*

Included in those scripts is `seldon-deploy-install/sd-setup/gitops-setup.sh`. The `Trial Installation` docs explain how to use it. Some of its steps read the `MULTITENANT` flag.

If using the `Product Installation` with gitops then this script should be consulted to adapt the steps for a multi-tenant cluster. The main differences are:

- The argocd installation is different as it uses different roles from the manifests that Argocd publishes.
- For each new namespace added to gitops, we have to add roles for Argocd to be able to sync it.
- For each new namespace a new instance of Seldon Core is installed, dedicated to that namespace.
- For each new namespace new roles are added for Seldon Deploy to be able to operate on that namespace.

The details of these differences can be found in the script.

Seldon Core

Seldon Core supports a namespaced install with a `singleNamespace` setting in its helm chart. There is an example in `seldon-deploy-install/sd-setup/gitops-setup.sh`.

KFServing

Only cluster-wide is currently supported. Also requires knative.

Minio

We suggest installing with the minio helm chart. By default we setup one instance per cluster but that could be configured and at the time of writing ClusterRoles are used only on OpenShift

Argo

We suggest using argo workflows default installation which is cluster-wide but namespaced installations are possible.

Note that each Seldon namespace that runs batch jobs would need the ability to start argo workflows. Batch jobs are optional.

Knative

At the time of writing neither knative serving nor knative eventing seems to officially support multi-tenancy. See below.

Configuring Seldon Deploy

Some functionality depends on knative. If knative is not used then this affects the request logger configuration. (Not using knative also means the outlier and drift detector demos won't work.)

Seldon Core has a default URL that its install instructions point to a knative broker in `seldon-logs` namespace (`executor.requestLogger.defaultEndpoint` for seldon core). This can be changed to point to `seldon-request-logger.seldon-logs` (or your request logger location).

Detector component wizards and the Data Science metrics component wizard do need knative 0.18.

The Seldon Deploy helm chart has a `rbac.clusterWide` setting. The `seldon-deploy-install/sd-setup/sd-install-default` script uses the `MULTITENANT` flag to set this.

There are two options from here.

Option 1 - One Deploy, per-Namespace RBAC

This is reduced RBAC rather than true multi-tenant.

If `rbac.readNamespaces` is true then a single Deploy instance should be used. This option requires a cluster-level read role to read namespaces.

When each namespace is added then new roles can be added for that namespace using the `apply-role-namespace.sh` or `gitops-setup.sh` script.

Note that the gitops script restarts Deploy each time a namespace is added.

Option 2 - Deploy Instance Per Namespace

This is full multi-tenant using only namespaced roles.

If `rbac.readNamespaces` is false then a Deploy per namespace should be used. Each Deploy then looks only at its own namespace.

The request logger namespace is configurable. This could be a dedicated namespace per Deploy instance or the same namespace as Deploy.

3.7.6 OpenShift Installation

Installations for OpenShift

3.7.6.1 Red Hat Marketplace Installation

Seldon Deploy is available in Red Hat markeplace with options for trial or purchase.

In Red Hat marketplace installation is via an operator. That operator needs to be installed first. It is installed using the marketplace tile.

The Red Hat marketplace installation is therefore different to other installations.

The marketplace installation is also different to other openshift installations. This is because all dependencies used in the installation are versions provided or certified by Red Hat.

To use the Red Hat marketplace installation, first choose the tile in your marketplace-enabled openshift cluster. Then follow the installation steps

Troubleshooting steps are included at the bottom of the install guide

3.7.6.2 Other OpenShift Installations

Red Hat marketplace is not the only way to install on OpenShift. Each of the components can be installed on OpenShift in essentially the usual way. You are likely to hit errors such as:

```
spec.containers[0].securityContext.runAsUser: Invalid value: 65534: must be in the␣
↪ranges: [1000620000, 1000629999]
```

These can be addressed by adding security context contstraints.

For example, if you hit this with seldon-core-analytics in the seldon-system namespace then run:

```
oc adm policy add-scc-to-group anyuid system:serviceaccounts:seldon-system
```

And similarly for kfserving and others.

This kind of OpenShift installation is not documented in detail but is not fundamentally different from other installs. Consult the docs of the providers of individual components and contact seldon in the case of issues.

3.7.7 Installation Types

Customers can install in multiple ways and need to choose when to use a Trial install or a Production install. So we need to ask:

- What is the difference between the Trial and Production Installations?
- Can a Trial setup not be used for Production?

Let's answer these questions.

3.7.7.1 Trial vs Production Install

The trial install is really a scripted installer that installs all components. Many organisations prefer to install each component individually (which we call the Production/Modular Install approach). The reasons are:

- Many organisations have existing installs for oidc/identity, elastic and sometimes other tools.
- Sometimes access to dockerhub and other public hosting is blocked, meaning each image has to be copied and checked in the org's hosting.
- The trial install is opinionated about versions - sometimes organisations have approved or preferred versions.
- The trial install makes assumptions about volume sizes and resource allocations.
- The trial install includes some components which are optional.
- The trial setup for https uses self-signed certs and many organisations have certs that they prefer to use.
- The gitops part of the trial install is optional but if used it assumes public GitHub.

The trial install can certainly be used at scale and is supported. It is just not customised to the organisation or the use case.

3.7.7.2 Easiest Ways to Get Started

The KIND setup is very quick to get running. It does not require a cloud provider account or any git repo access.

The next easiest option is the trial install. We suggest to setup without gitops first and then add gitops after (the trial setup has gitops as a separate script).

If moving to a production installation we suggest to add components progressively. In a minimal setup deploy can be run without auth or gitops and this helps simplify initial configuration so that each step can be verified. Our docs also include verification and troubleshooting steps for each component.

3.7.7.3 What if I also want an ML training platform?

Seldon tools can work with many ways of training models. Some users choose to train models in CI systems or notebooks. There's also a range of ML training platforms such as kubeflow or mlflow.

We have a special relationship with kubeflow as major contributors. We see many customers interested in kubeflow so we provide an install option that includes kubeflow.

Other platforms can certainly be used. Seldon Deploy can work with any way of building and installing Seldon core models. Seldon core users make use of a wide range of platforms, including cloud provider offerings like sagemaker, as well as open source tools like kubeflow.

3.7.7.4 How to Size a Cluster?

The default install makes sizing assumptions for each of the components. But sizing varies with use.

Some customers will make more requests and put more load on elasticsearch. Others will have more users logged in viewing monitoring, which will mean more prometheus queries.

Sizes can be adjusted with kubernetes. Even with the trial install, the scripts are provided and resource values can be adjusted and reapplied. If cluster limits are reached then with Kubernetes nodes can be added to the cluster while it is running.

CHAPTER

FOUR

PRODUCT TOUR

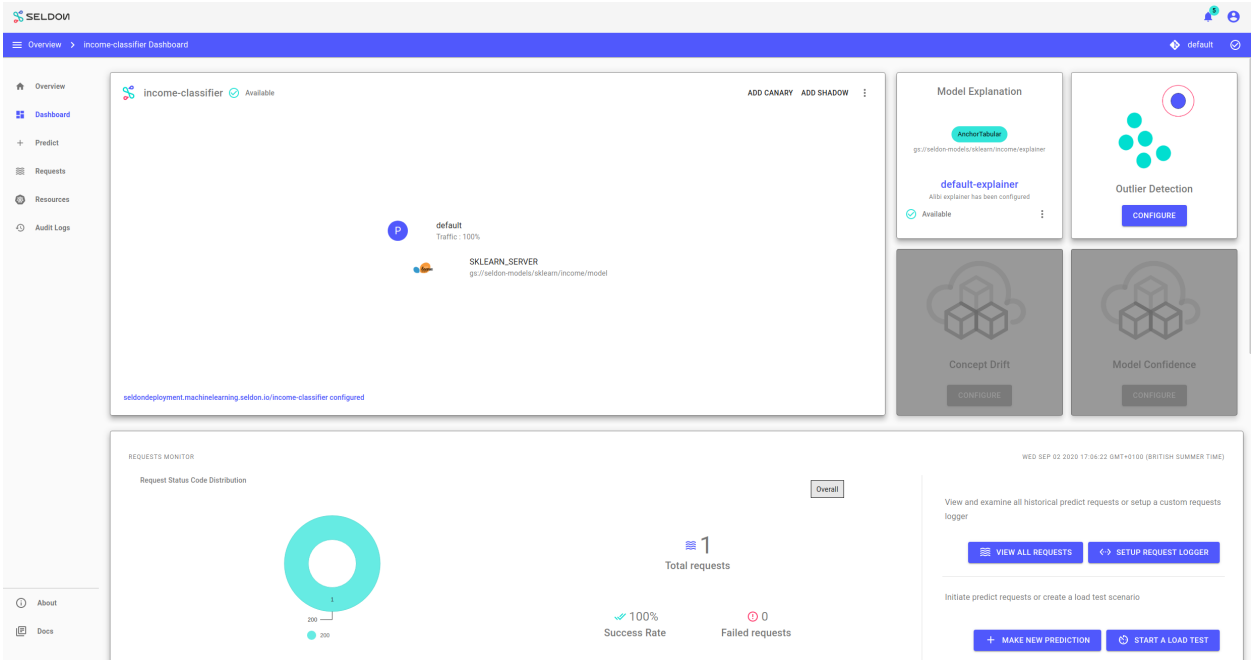
Walkthrough of Seldon Deploy features

A tour of Seldon Deploy features.

4.1 Deployment Dashboard

Understand the running machine learning model and do core operations

Deployment Details provides a summary for a model.

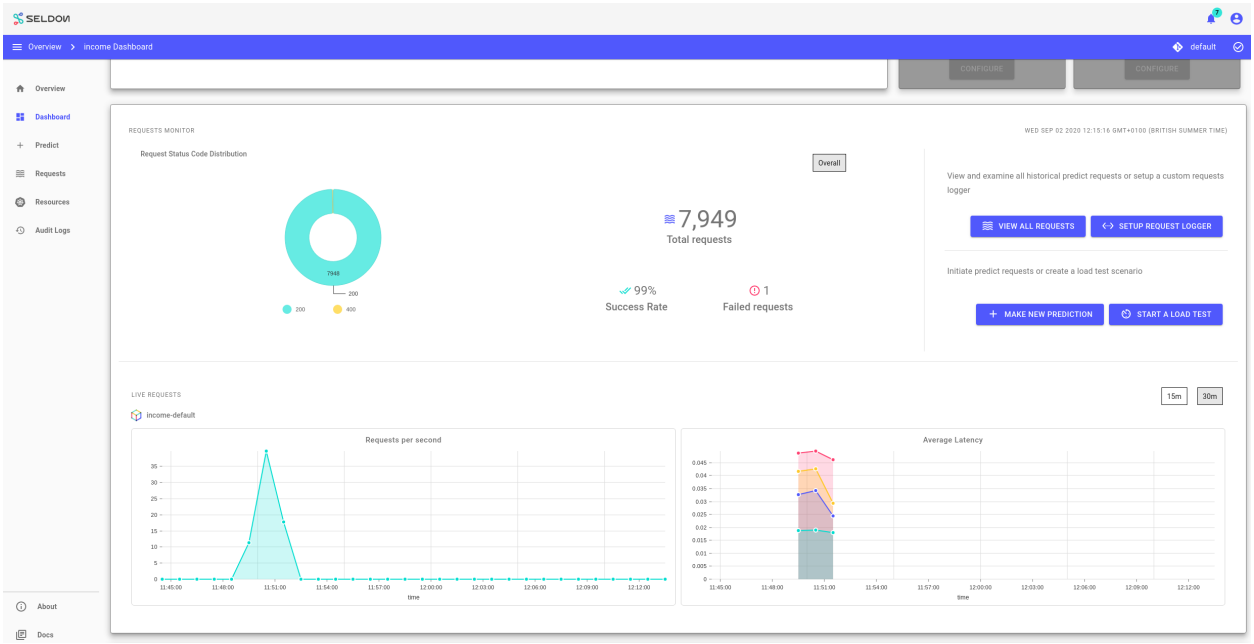


The page also provides a starting point for inspecting and performing actions on the deployment.

The main tile shows the model/models in the deployment. It has a drop-down action menu from which the deployments full descriptor or audit log (see `GitOps` under `Deployments by Namespace`) can be accessed, or the model can be updated (using one of the `Rollout Strategies`) or deleted.

4.1.1 Request Monitoring

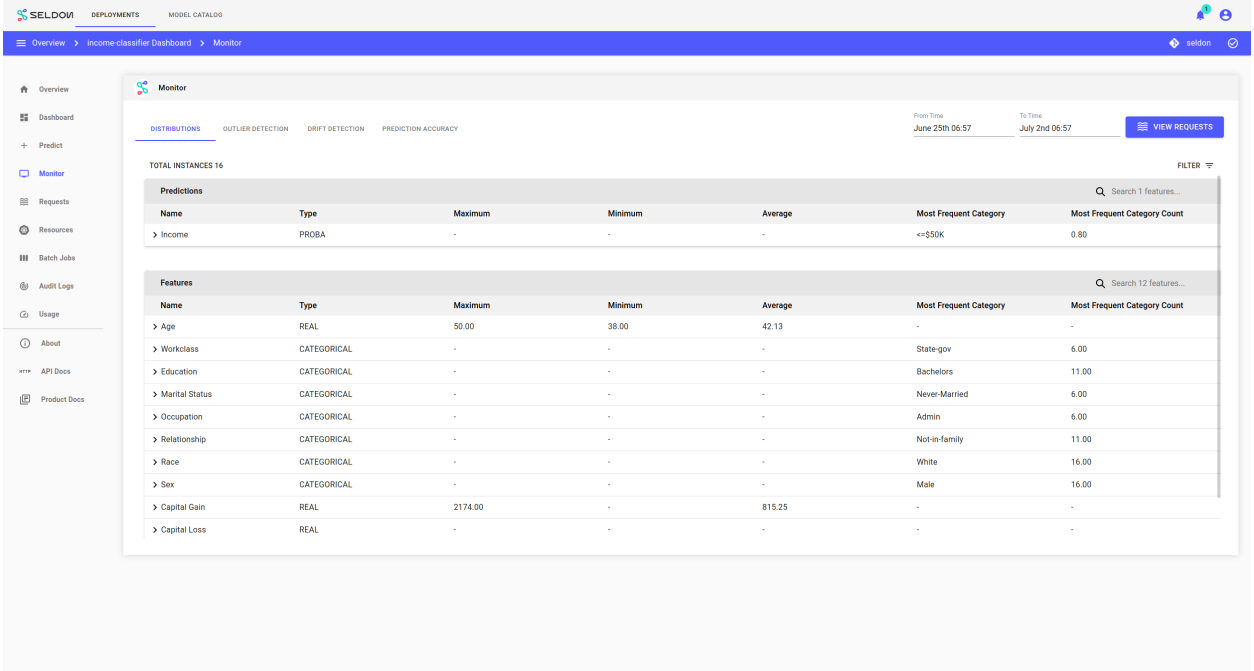
The deployment dashboard view includes request monitoring panels to view real-time predictions metrics



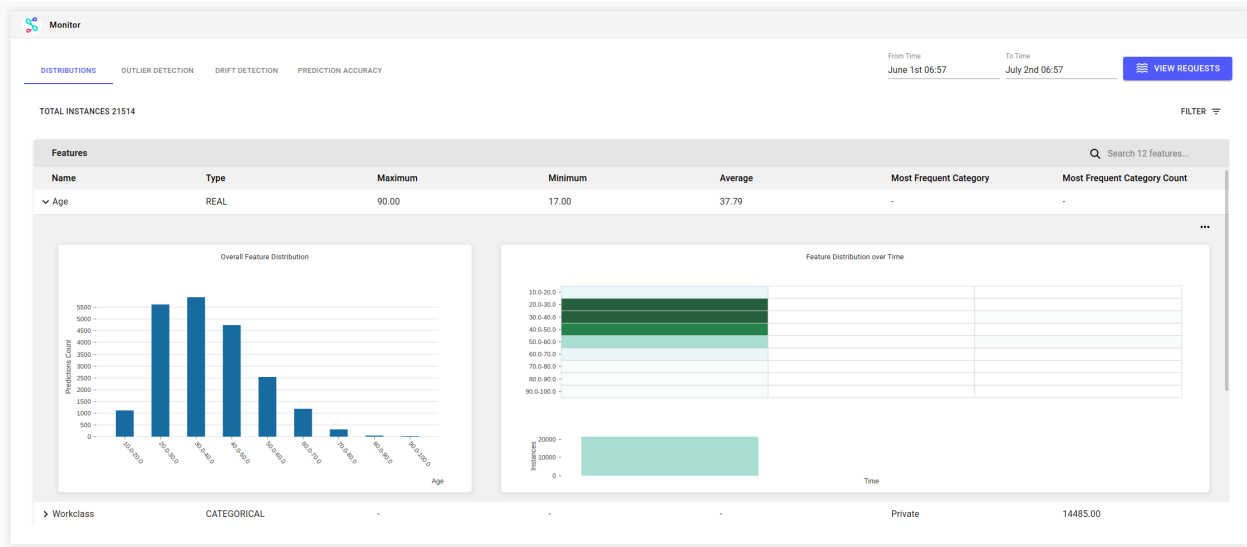
These show a summary of traffic going through the model, including request rate, indicators of request success/failure (based on status code) and latency.

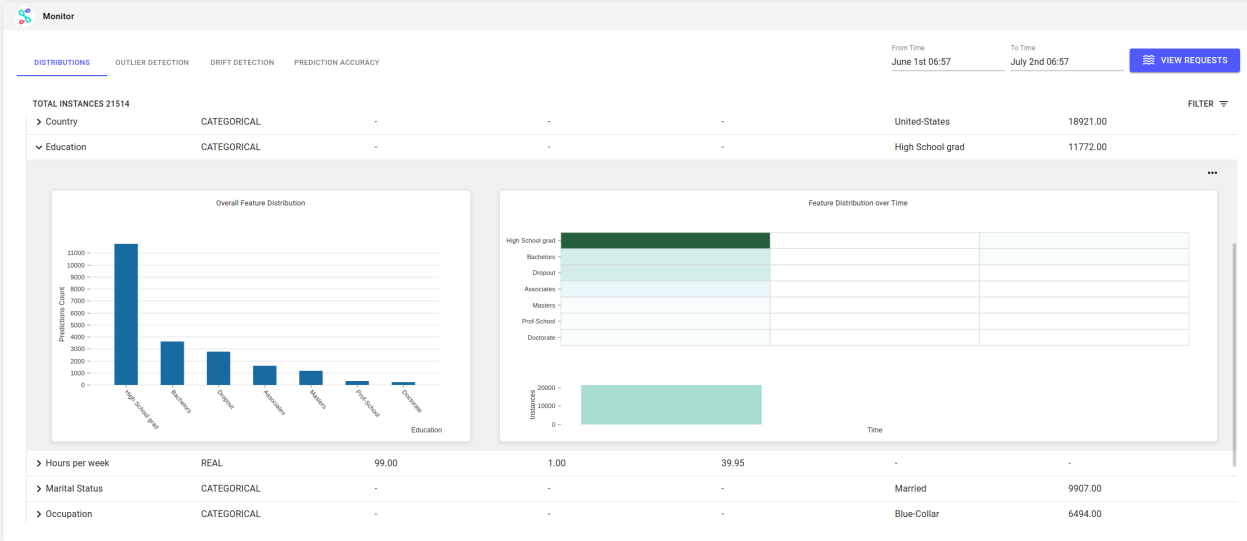
4.1.2 Distributions Monitoring

It is a vital aspect of model monitoring cycle to understand if the deployed model has the desired prediction characteristics during different times and for different cohorts. Distributions monitoring dashboard provides an ability to view the statistics and distributions of features and predictions made by your model between any given time.



This feature also enables you to draw comparisons between the model predictions for different feature combinations, cohorts and/or time slices by choosing appropriate filters.





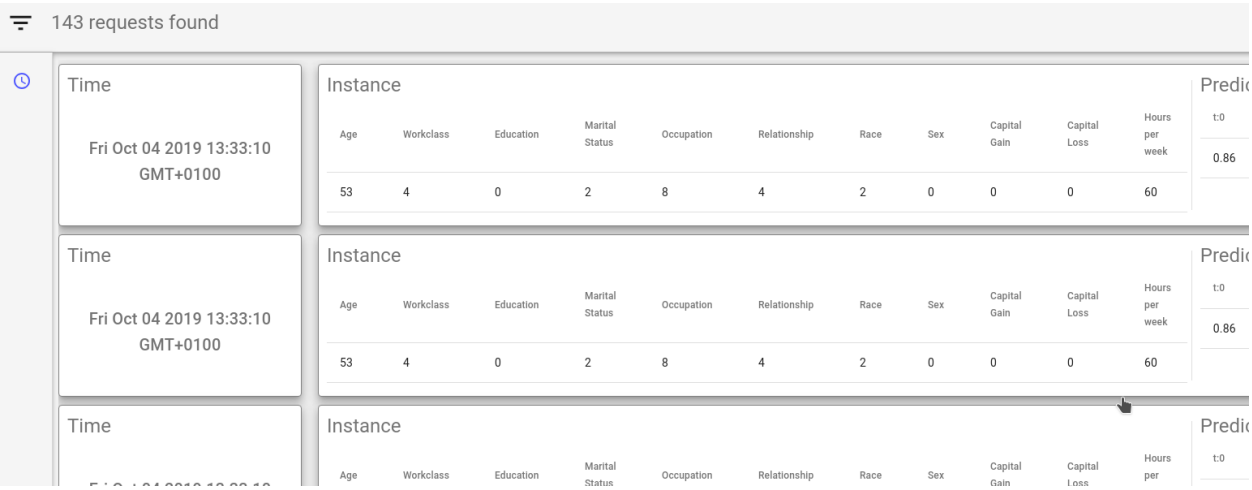
Relevant Demos

- *Seldon Core Deployment Demo*
- *Inference Service Deployment Demo*

4.1.3 Request Logs

Requests can be made using the prediction or load test options in the UI or an authorised request directly to the model.

If the model is a SeldonDeployment then we step into the request in the request log page:



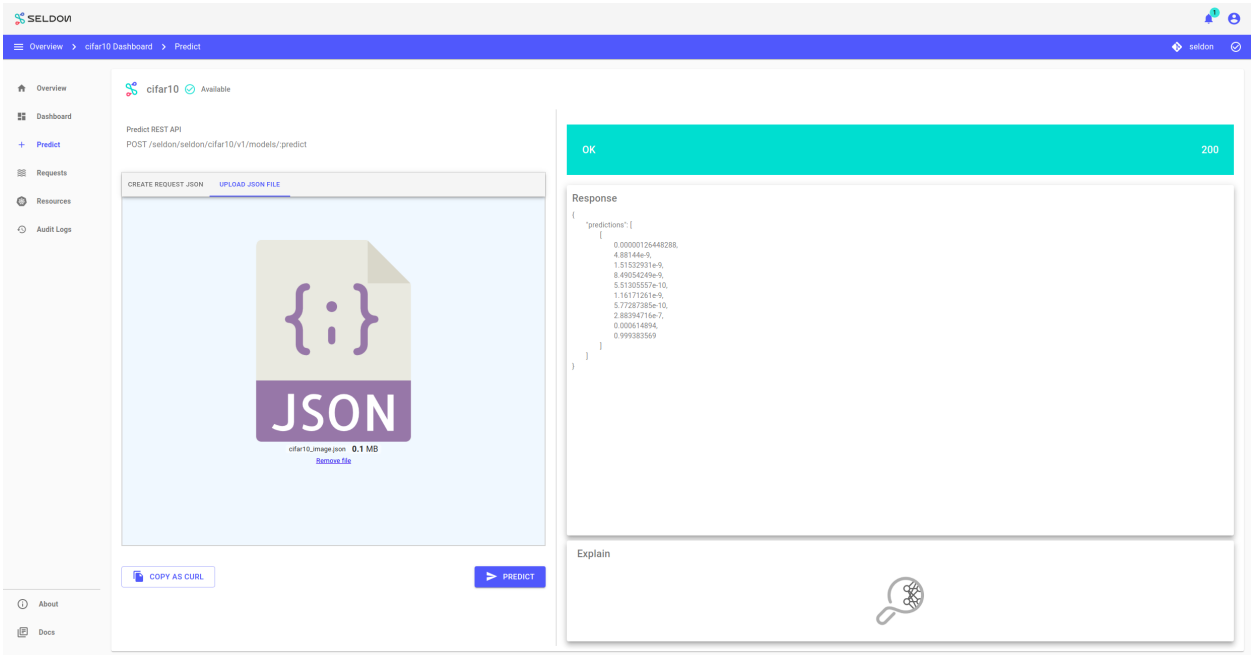
See the architecture sections for more on the Seldon Core use of the EFK stack.

4.1.4 Requests to Models

Requests can be made using the user interface via different options.

4.1.4.1 Make a prediction

The prediction can be made from the UI directly by pasting or uploading content. The response is then shown on the screen.

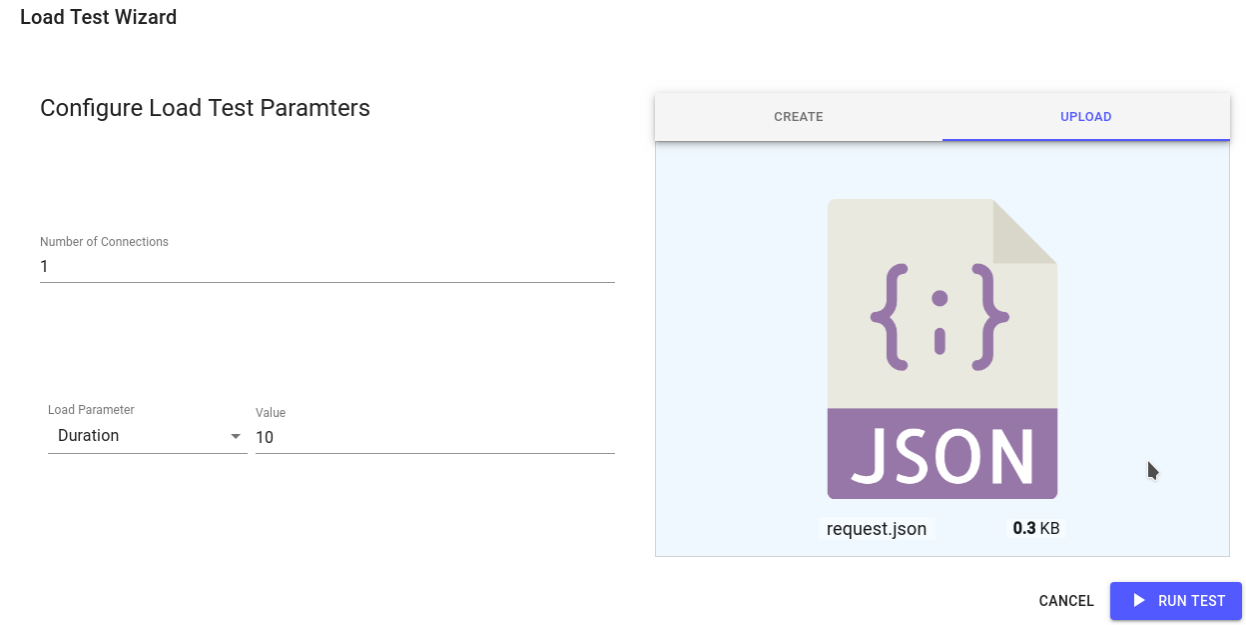


Here the path to model is displayed and an option is provided to export a curl command for manual requests.

SeldonDeployment models default to the Seldon protocol and URL form. Alternatively, the tensorflow protocol can be used and then Deploy will infer a tensorflow URL. With Seldon the model name needs to be specified as a parameter in the manifest. Content of tensorflow requests is different as explained in a seldon core notebook.

4.1.4.2 Load Test

This initiates a loadtest, which in the background is implemented using hey and exposes the same options as that tool

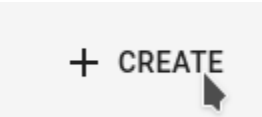


The load test runs inside the cluster so can take time to be provisioned.

4.2 Deployment Wizard

Create new deployments with simple user-friendly wizards

To simplify the creation of SeldonDeployment and KFService manifests, Seldon Deploy provides a `Create` button:

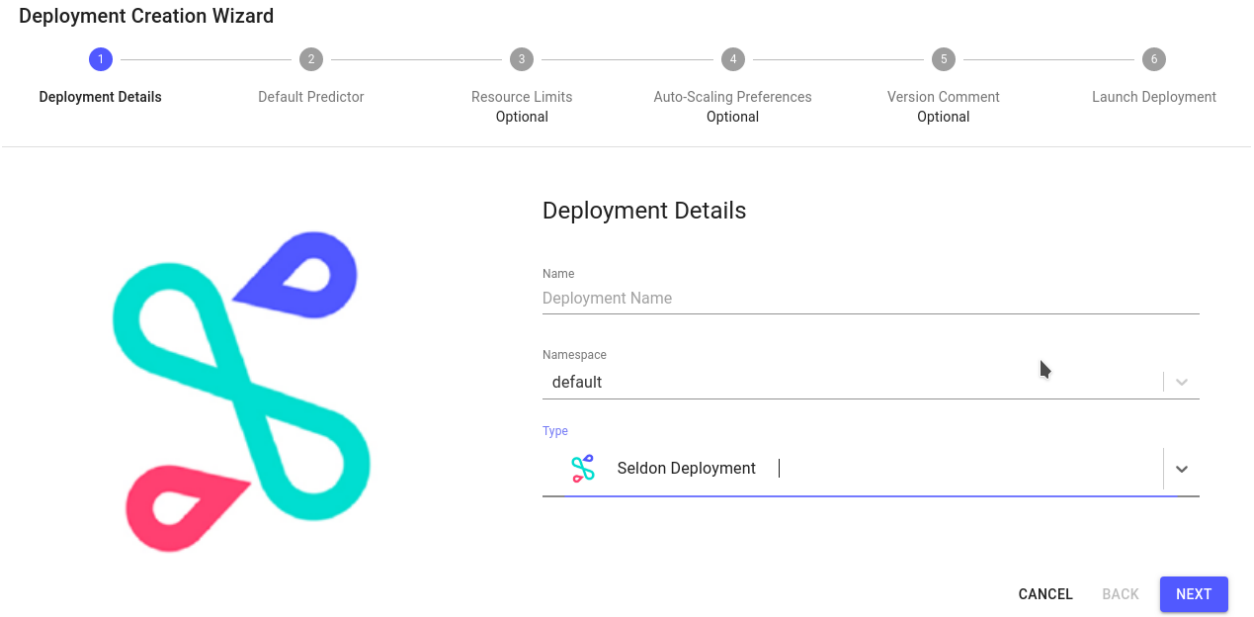


This launches a wizard to step through simple manifest creation. The wizard will include options to choose toolkit (e.g. tensorflow) and location of the model file (pyton pickle).

For more on how the location is specified, see the Seldon Core or the KFServing examples

4.2.1 Seldon Deployment Wizard

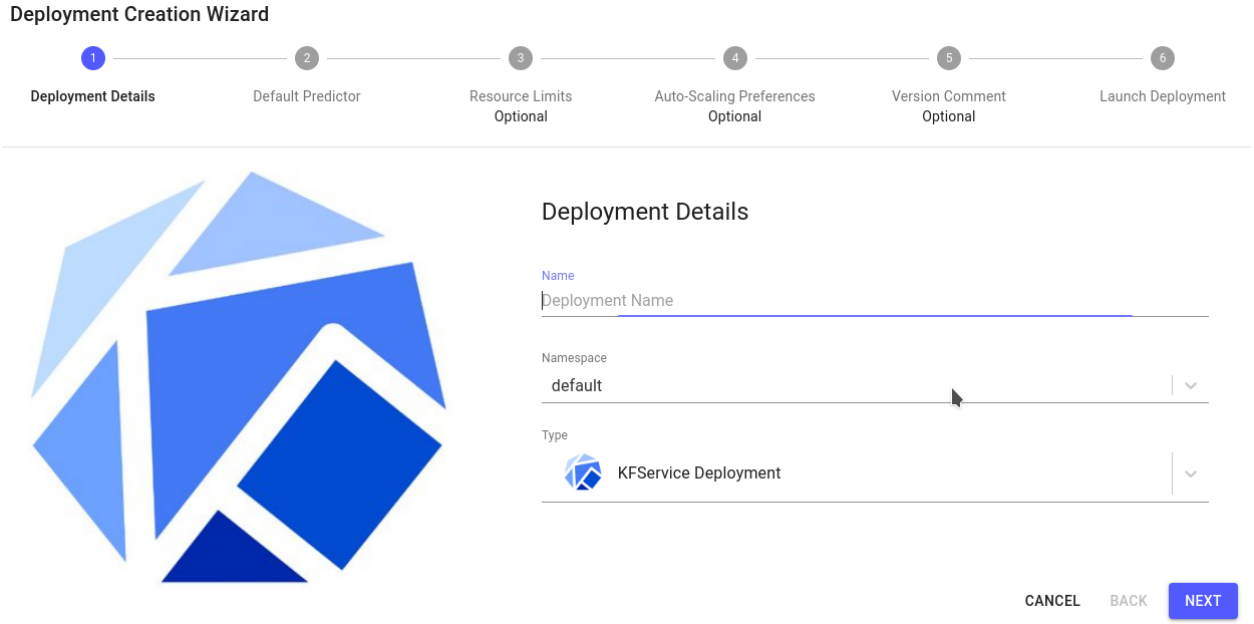
The SeldonDeployment wizard is used if the user chooses a KFService after using the Create button



This differs from the KFServing wizard in its autoscaling options. If enabled (disabled by default), the SeldonDeployment options are for the HorizontalPodAutoscaler. This allows for scaling up and down pod numbers based on e.g. cpu utilisation.

4.2.2 Kubeflow Inference Service Wizard

The KFService wizard is used if the user chooses a KFService after using the Create button



This differs from the SeldonDeployment wizard in its autoscaling options. A minimum and maximum number of pods can be set. This is in accordance with KFServing's scaling based on knative, in response to the rate of requests.

4.3 Model Explanations

Understand why models give particular predictions

Model explanations allow you to understand why a machine learning model gave a particular predictions. Seldon Deploy allows you to use Black Box Model Explainers that can be used on models built using any technology.

4.3.1 Available Methods

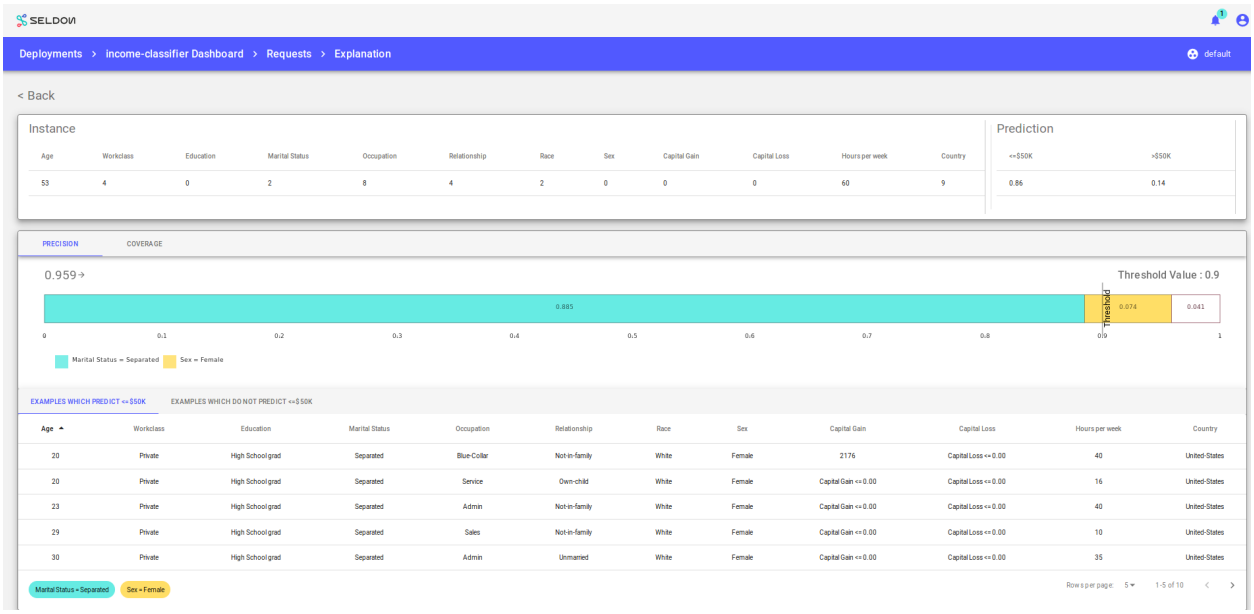
Seldon Deploy supports a subset of the methods currently available in Alibi for both Seldon Core and KFServing deployments. Find documentation on Anchor explanation algorithm and read about creating your custom explainer

Explanation Method	Seldon Core	KFServing	Alibi Docs
Anchor Tabular	*Example*	*Example*	Docs
Anchor Text	*Example*	*Example*	Docs
Anchor Images	*Example*	*Example*	Docs

4.3.1.1 Tabular Data

Explainer model classifier with tabular features

For explanations for tabular datasets predictions can be explained using techniques such as Anchors Tabular to provide an understanding of the features the model is using to make it prediction.



We provide an *anchor tabular explainer demo* with Seldon Core.

Details about the methodology can be found in the alibi documentation

4.3.1.2 Text Data

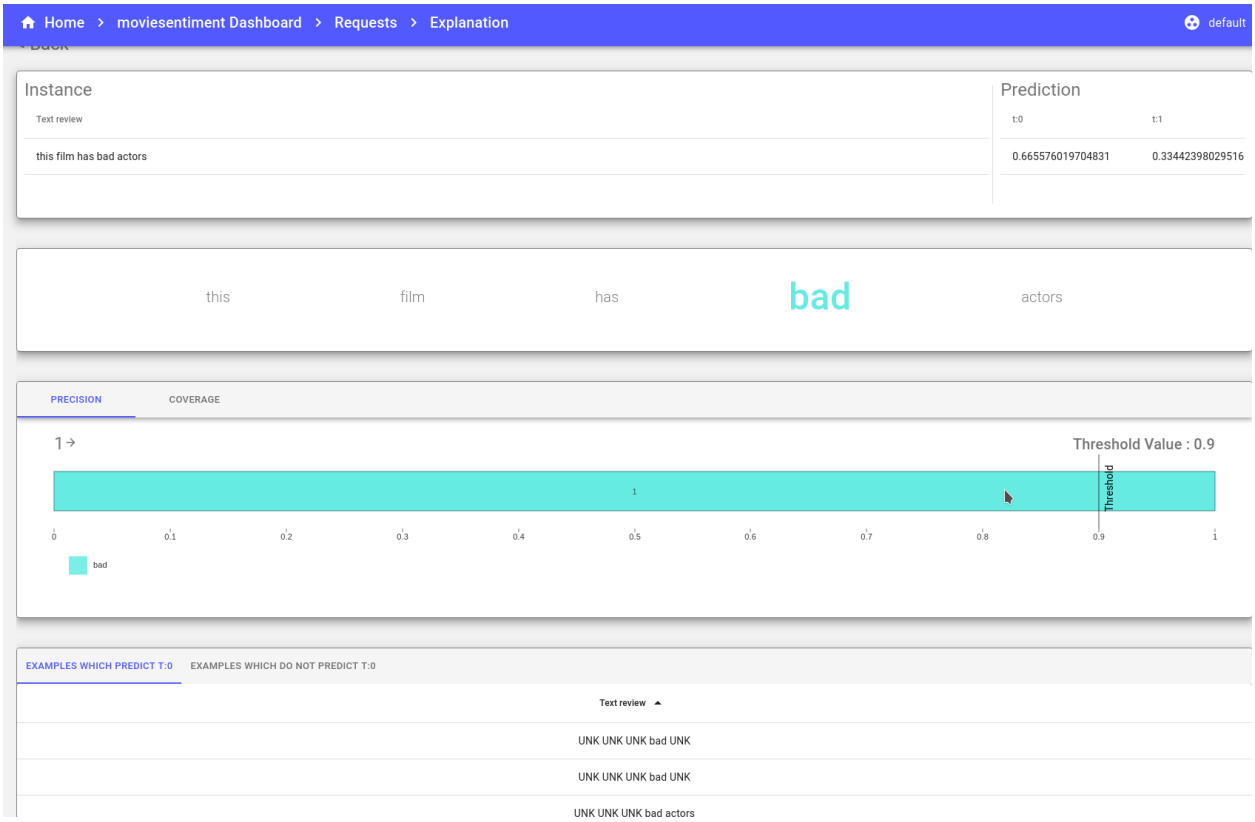
Explain models whose input is textual

A good way to see text expalantions in Deploy is to take the movie review example from the Seldon Core documentation. The example there makes the request using the seldon client but it can also be made from the Seldon Deploy UI with the following JSON:

```
{
    "data": {
    "names": ["Text review"],
    "ndarray":
    ["this film has bad actors"]

    }
}
```

The Seldon Deploy UI can then show a visual representation of the explanation:



This clearly highlights that the word 'bad' is considered negative and that influenced the classification of the review as negative.

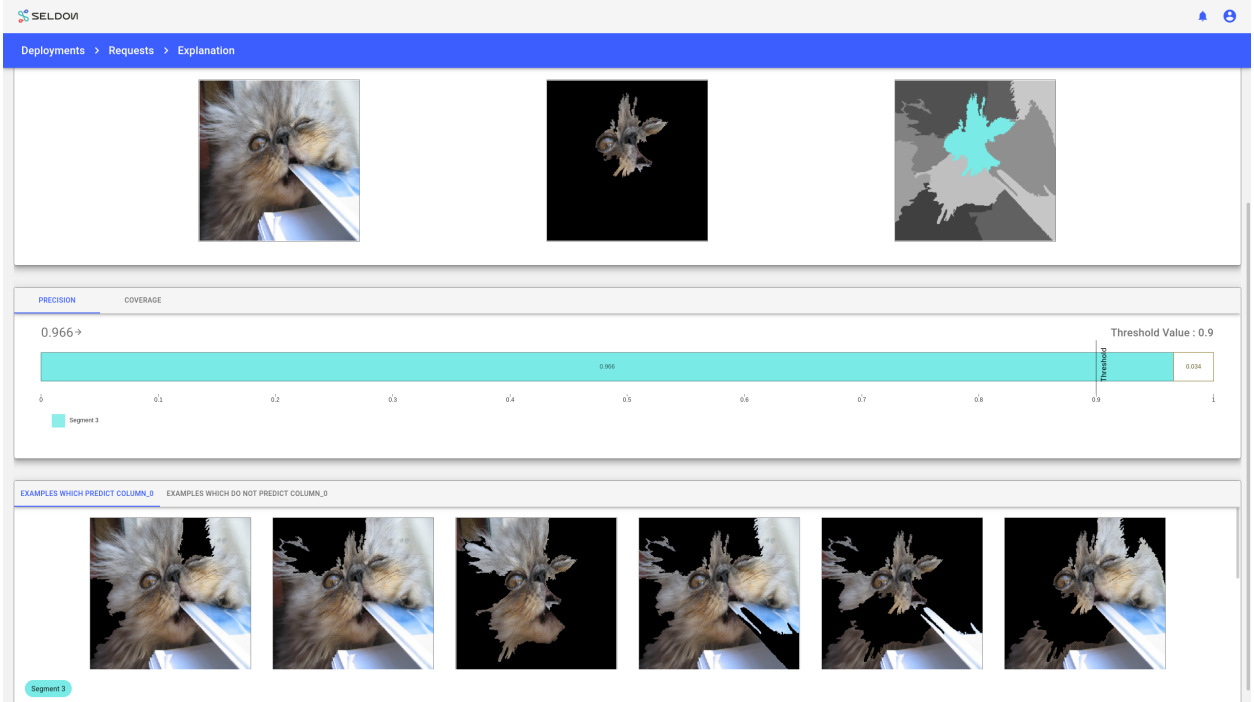
We provide an *anchor text explainer demo* with Seldon Core.

More can be seen about the methodology in the alibi documentation

4.3.1.3 Image Data

Explain image based classifiers

For explanations for image datasets predictions can be explained using techniques such as Anchors Image to provide an understanding of the segments of the image that leads to a specific prediction.



Details about the methodology can be found in the alibi documentation

4.4 Rollout Strategies

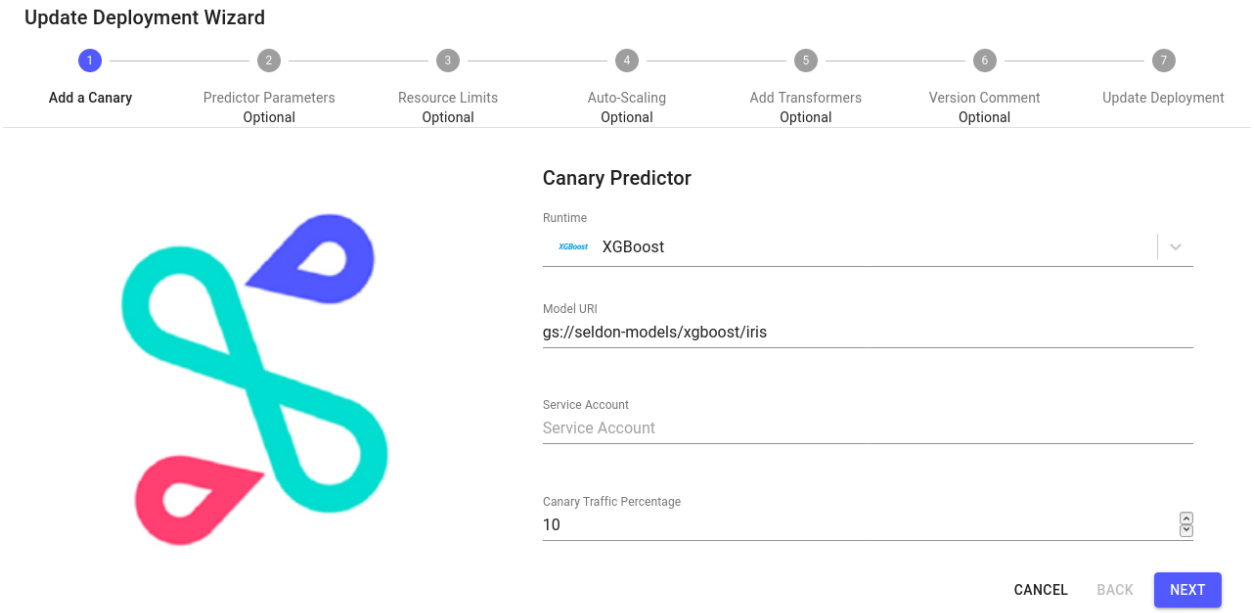
Rollout strategies for ML Models

The typical rollout strategy for kubernetes is a rolling update, where traffic is moved over from a live version to a new version when the new version is ready. For Machine Learning it is common to perform more complex rollouts where traffic is split between versions of a model in a customized way.

4.4.1 Canary

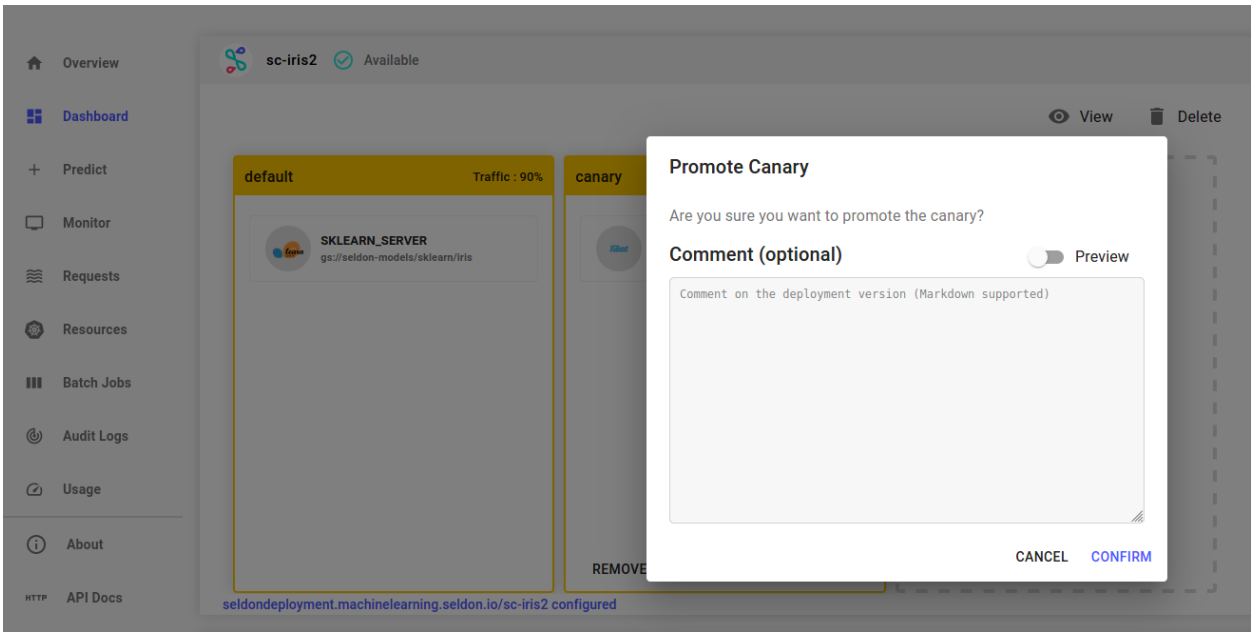
With a canary rollout traffic is split between a main model, which receives a majority of traffic, and a new version, which is given a fraction of traffic until it is decided whether to promote the model. The features for this can be seen in full in the Demos section of the documentation under `Deploying, Load Testing and Canarying Seldon Core Servers`. The key features in particular are:

1. A wizard to add a canary:



1. Visualizing metrics for both default and canary models:

1. Promotion of the Canary to be the main model.

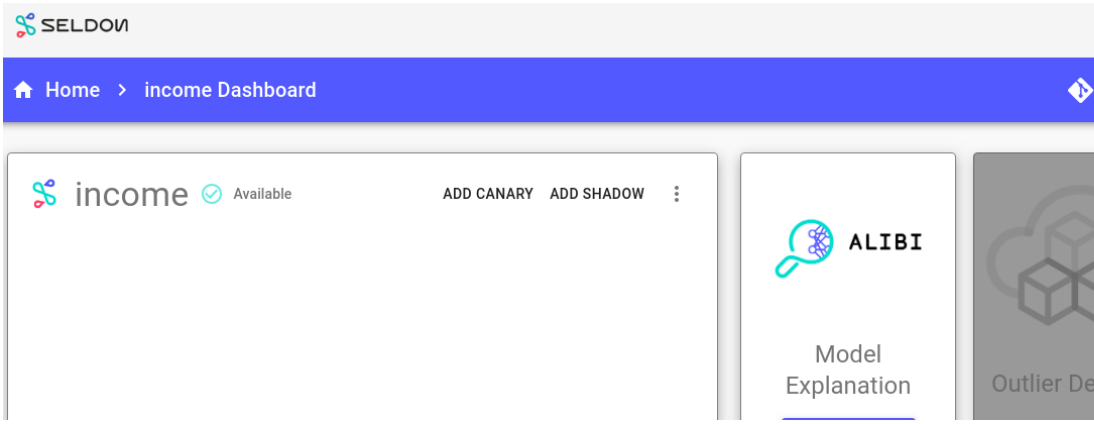


1. GitOps integration for the whole process, so that all changes can be audited (see GitOps under Architecture)
2. Request logs include any requests that go to the canary, with the responses.

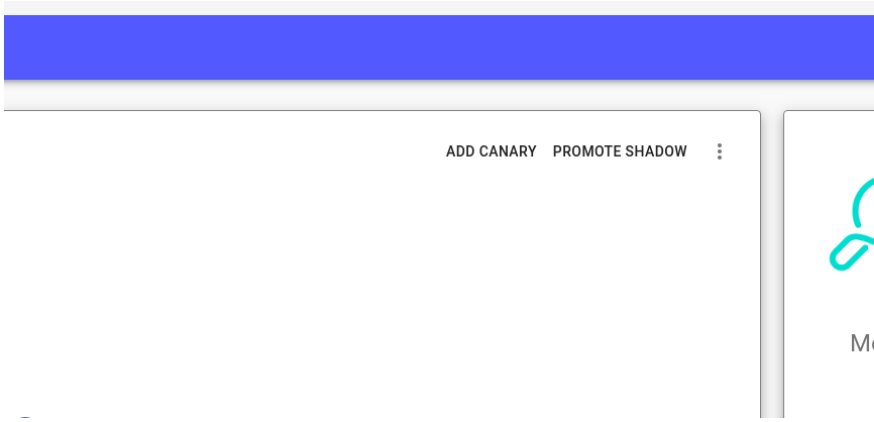
4.4.2 Shadow

With a shadow traffic is duplicated so that it goes to both a main model and also a shadow model. The shadow is typically a newer version being tried out. The responses to requests that reach the end user are only from the main model, not the shadow. But the shadow requests can be logged and the shadow can be monitored. Within Seldon Deploy there are features to:

1. A wizard to add a shadow:



2. Visualizing metrics for both default and shadow models:



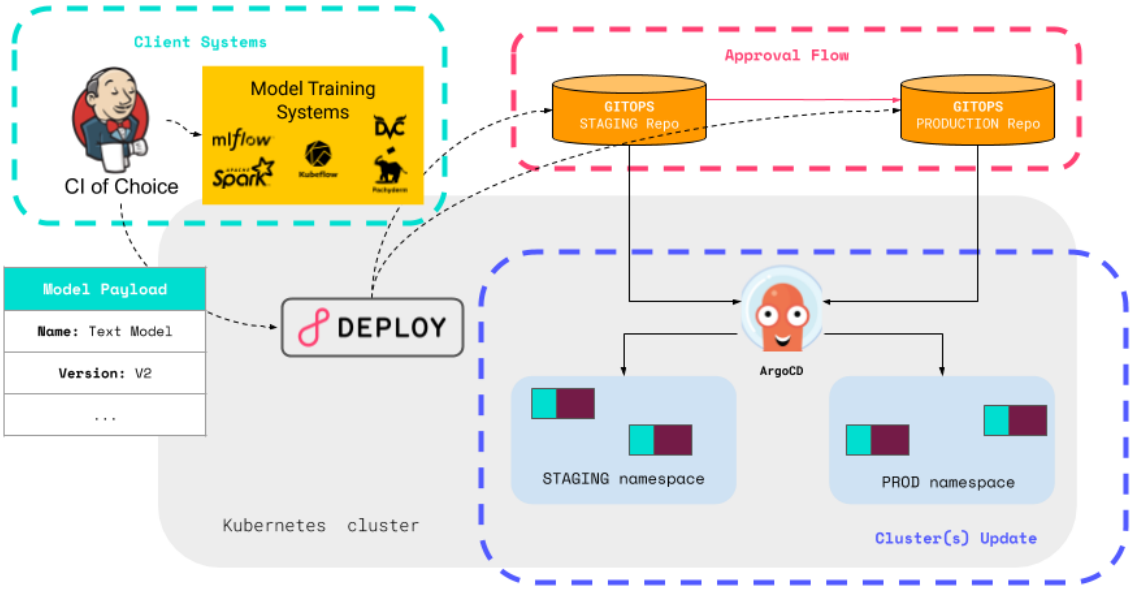
3. Promotion of the Shadow to be the main model.
4. GitOps integration for the whole process, so that all changes can be audited (see GitOps under Architecture)
5. Request logs include any request to the shadow, with the responses.

4.5 REST API

Seldon Deploy API

> **Warning:** Seldon Deploy's REST API is considered **experimental**. The following endpoints could change at any time and breaking changes are expected.

The REST API of Seldon Deploy lets you to interact with your machine learning deployments programmatically. This allows you to build complex deployment pipelines to integrate Seldon Deploy with any upstream services.



If you already have access to a Seldon Deploy installation, you can visit the **interactive API reference** to learn more about the Seldon Deploy API and the endpoints that it exposes. The interactive API documentation can be accessed by clicking on your profile icon, and then clicking on `API Docs`. Alternatively, you can go directly to `$ML_PLATFORM_HOST/seldon-deploy/swagger/`, where `$ML_PLATFORM_HOST` needs to be replaced by the domain where your Seldon Deploy installation can be accessed.

4.5.1 Usage

The recommended way of interacting with the Seldon Deploy REST API is through its Python SDK. However, you can also use plain `cURL` to send requests.

You can find some example usages below.

Note: These usage examples assume that you have already obtained an access token following the instructions of the Authentication section.

---

cURL

Python

We can use `cURL` (available on most distributions) or similar HTTP clients to interact directly with the Seldon Deploy API.

For example, if we assume that there is an authentication token present in the `$TOKEN` variable, we could list our machine learning deployments as:

```
ML_PLATFORM_HOST="https://ml.example.com"
curl -k -X GET \
  "$ML_PLATFORM_HOST/seldon-deploy/api/v1alpha1/namespaces/staging/seldondeployments"␣
↪\
  -H "Authorization: Bearer $TOKEN"
```

To use the Python SDK, the first step will be to install the `seldon-deploy-sdk` package. You can do so using `pip` as:

```
pip install seldon-deploy-sdk
```

Once we have obtained an authentication token, we will need to set it as the `access_token` of our `Configuration` object. Afterwards, we could list all our machine learning deployments under the `staging` namespace as:

```
from seldon_deploy_sdk import Configuration, ApiClient, SeldonDeploymentsApi

config = Configuration()
config.host = "http://ml.example.com/seldon-deploy/api/v1alpha1"
config.access_token = "<AUTH_TOKEN>"

api_client = ApiClient(config)

# List current machine learning deployments
sdep_api = SeldonDeploymentsApi(api_client)
sdeps = sdep_api.list_seldon_deployments("staging")

print(sdeps)
```

You can find more details on the Python SDK reference documentation.

4.5.2 Authentication

All requests to the Seldon Deploy API must be authenticated. Therefore, before using the API you must obtain an authentication token. Note that the process to issue a new authentication token may change depending on your architecture and your OIDC provider.

You can see some authentication examples below.

OIDC Client

In general, to authenticate against an OIDC provider, we will assume that the **`password` flow** is supported and that the OIDC client is **public**.

In Keycloak, this can be set by enabling `Direct Access Grants` and setting the client as `open` through the client dashboard in the admin UI.

---

cURL

Python

We can use plain `cURL` to obtain a token, by emulating OpenID's `password` flow.

If we assume a set up where Keycloak is configured as an OIDC provider and that there is an OpenID client named `sd-api`, we could obtain an authorization token to access the API using plain `cURL` as:

```
SD_USER="data-scientist-1@example.com"
SD_PASSWORD="12341234"
CLIENT_ID="sd-api"

KEYCLOAK_HOST="https://auth.example.com"
KEYCLOAK_REALM="deploy-realm"

```

(continues on next page)

(continued from previous page)

```
_payload="username=$SD_USER&password=$SD_PASSWORD&grant_type=password&client_id=
↪$CLIENT_ID&scope=openid"
_authEndpoint="$KEYCLOAK_HOST/auth/realms/$KEYCLOAK_REALM/protocol/openid-connect/
↪token"
RESULT=$(curl -k -s --data $_payload $_authEndpoint)
TOKEN=$(echo $RESULT | sed 's/.*id_token":"//g' | sed 's/".*//g')

echo "TOKEN=$TOKEN"
```

Note that this example also assumes that the `sd-api` OpenID client is public and that it supports the `password` flow.

Out of the box, the Python SDK supports a set of common authentication workflows. Each of these, can be found under the `seldon_deploy_sdk.auth` package.

- `SessionAuthenticator`: Allows you to authenticate against Dex, configured as an ingress-level authentication provider. This is equivalent to **Seldon Deploy's** ***default Kubeflow setup***.
- `OIDCAuthenticator`: Allows you to authenticate against an OIDC-compatible provider, using the `password` flow. This is equivalent to **Seldon Deploy's** ***default trial setup***.

As an example, let's assume a set up where Keycloak is configured as an OIDC provider and there is an OpenID client named `sd-api`, under a realm named `deploy-realm`. Note that this architecture is **equivalent to Seldon Deploy's** ***default trial installation***. Under these conditions, we could obtain an access token, using the `seldon_deploy_sdk.auth.OIDCAuthenticator` helper, as:

```
from seldon_deploy_sdk import Configuration
from seldon_deploy_sdk.auth import OIDCAuthenticator

sd_user = "data-scientist-1@example.com"
sd_password = "12341234"

config = Configuration()
config.host = "http://ml.example.com/seldon-deploy/api/v1alpha1"
config.oidc_client_id = "sd-api"
config.oidc_server = "http://ml.example.com/auth/realms/deploy-realm"
# For private OpenID clients:
# config.oidc_client_secret = "my-secret"

# Authenticate against an OIDC provider
auth = OIDCAuthenticator(config)
access_token = auth.authenticate(sd_user, sd_password)

# Configure the obtained access token as the one to use downstream
config.access_token = access_token
print(config.access_token)
```

You can find more details on the Python SDK reference documentation.

4.5.3 Versioning

The API endpoints are versioned to avoid clashes between different versions of the API. The current version of the API is `v1alpha1`, which means that breaking changes are still highly likely to happen. Once the current version *graduates* to stable, it will be renamed to `v1`.

Note that this versioning schema is similar to the one followed in Kubernetes.

4.5.4 Reference

4.5.4.1 API Reference (v1alpha1)

Reference for version v1alpha1 of the API

CHAPTER

FIVE

DEMOS

Important: These demos assume you have a running Seldon Deploy installed with relevant permissions to create/update deployments.

5.1 Seldon Core Demos

These demos assume you have a running Seldon Deploy installed with relevant permissions to create/update deployments.

They mostly use pre-built models, except the kubeflow demo which has steps to build a model and push it to minio.

Models can be pushed to minio or other objects stores for pre-packaged model servers or packaged as docker containers using language wrappers. See the Seldon Core docs for the building and hosting stages.

However you install your SeldonDeployments, Seldon Deploy should see them in visible namespaces. These demos use the Deploy UI for ease.

5.1.1 Canary Promotion

5.1.1.1 Iris Model

We will:

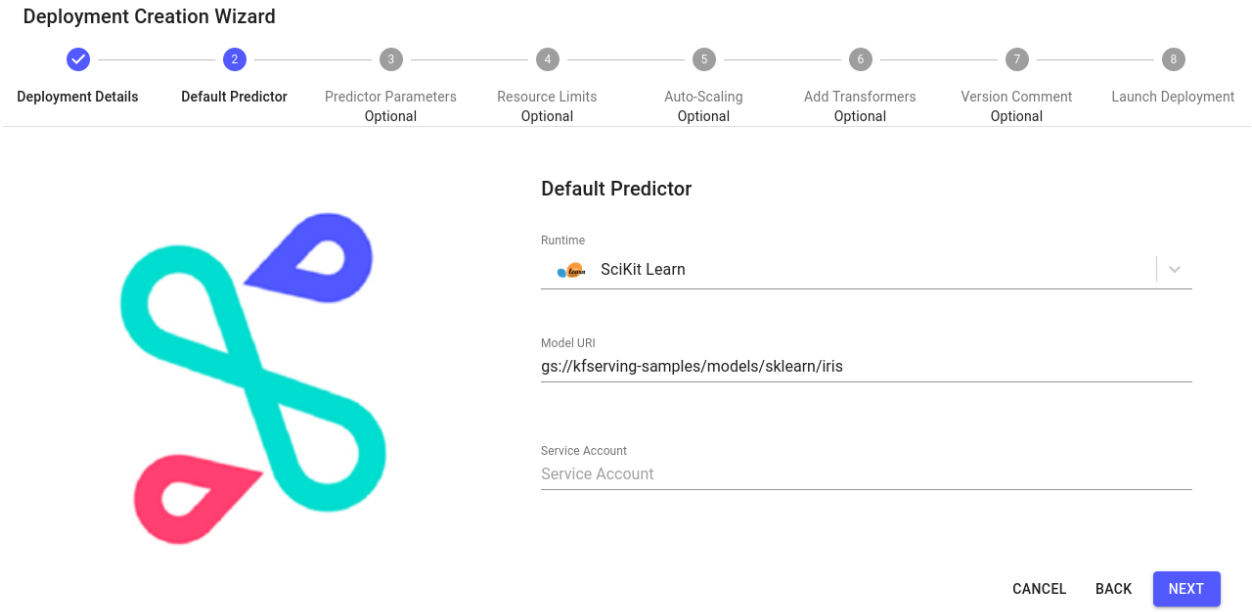
- Deploy a pretrained sklearn iris model
- Load test the model
- View request payloads
- Canary a new XGBoost model
- Load test canary model
- Promote the canary model

5.1.1.2 Deploy Model

Create the model using the wizard with a name of your choice and the model uri:

```
gs://seldon-models/sklearn/iris
```

All other defaults can be left as provided.

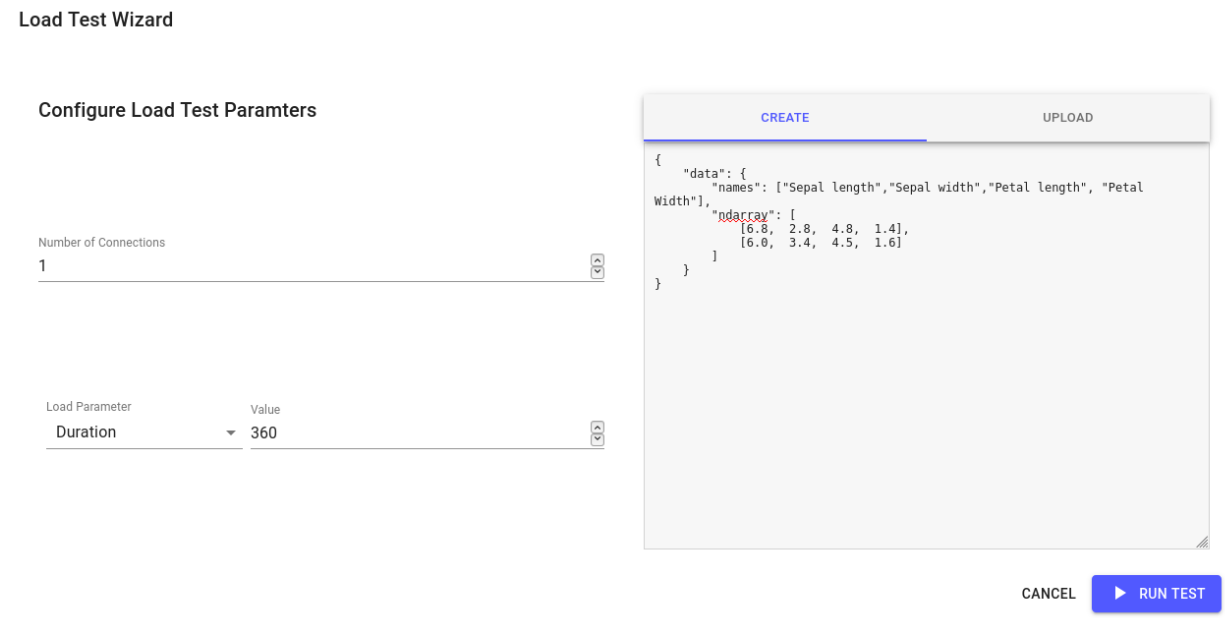


5.1.1.3 Start Load Test

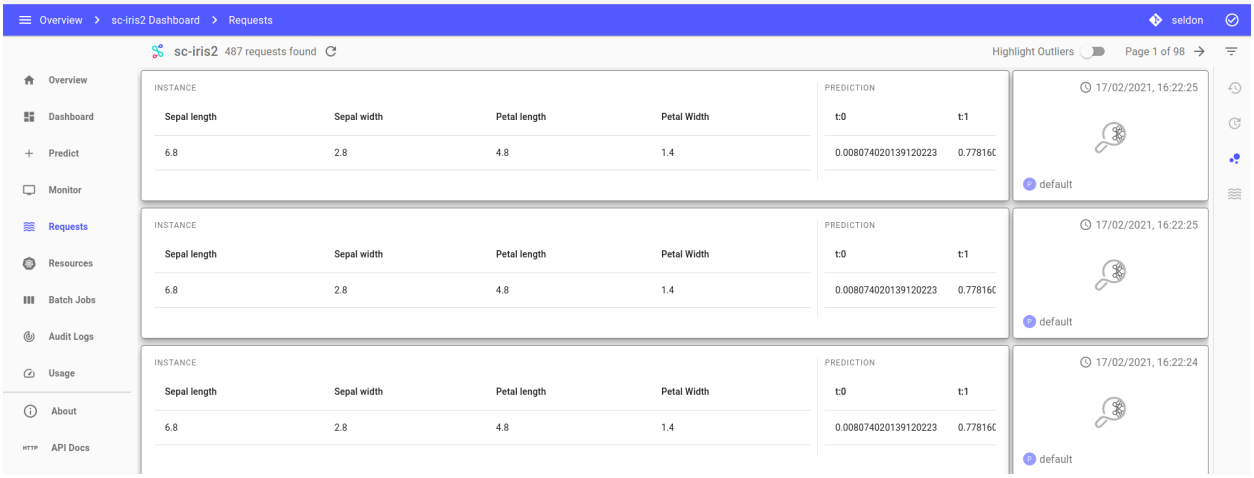
One the model is running start a load test with the following request payload:

Use the `request.json` file in this folder:

```
{
    "data": {
        "names": ["Sepal length","Sepal width","Petal length", "Petal Width"],
        "ndarray": [
            [6.8,  2.8,  4.8,  1.4],
            [6.1,  3.4,  4.5,  1.6]
        ]
    }
}
```



When running you should see *metrics* on dashboard. Enter the *request logs* screen to view requests. If this doesn't work, consult the *metrics* or *request logging* docs section for debugging.

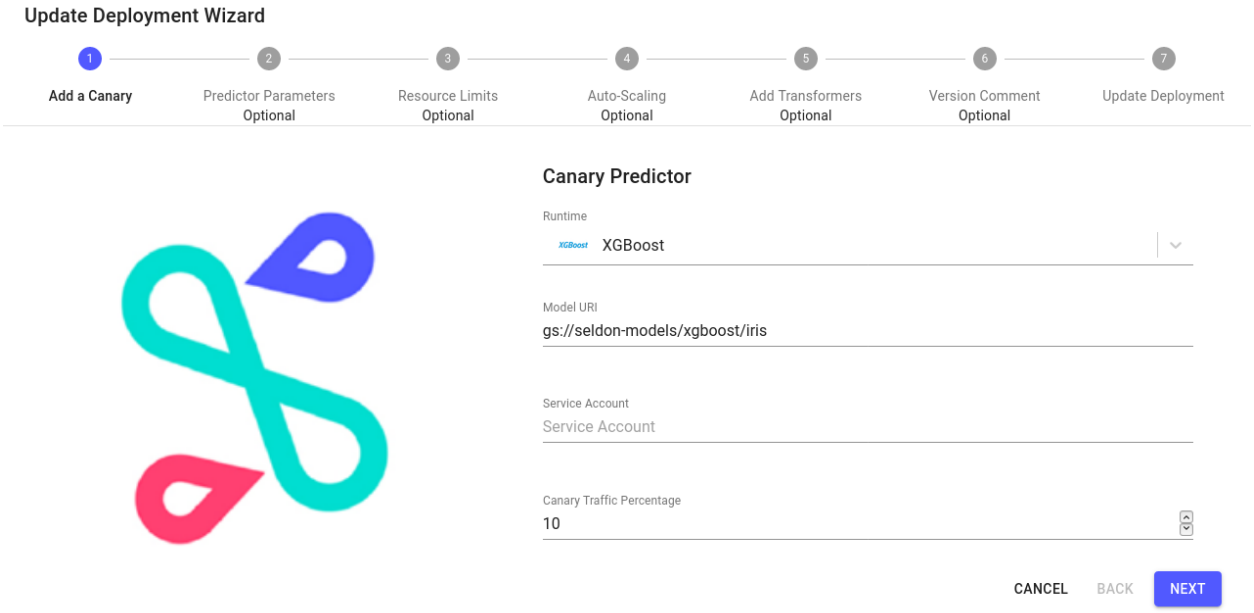


You can also see core metrics from the dashboard.

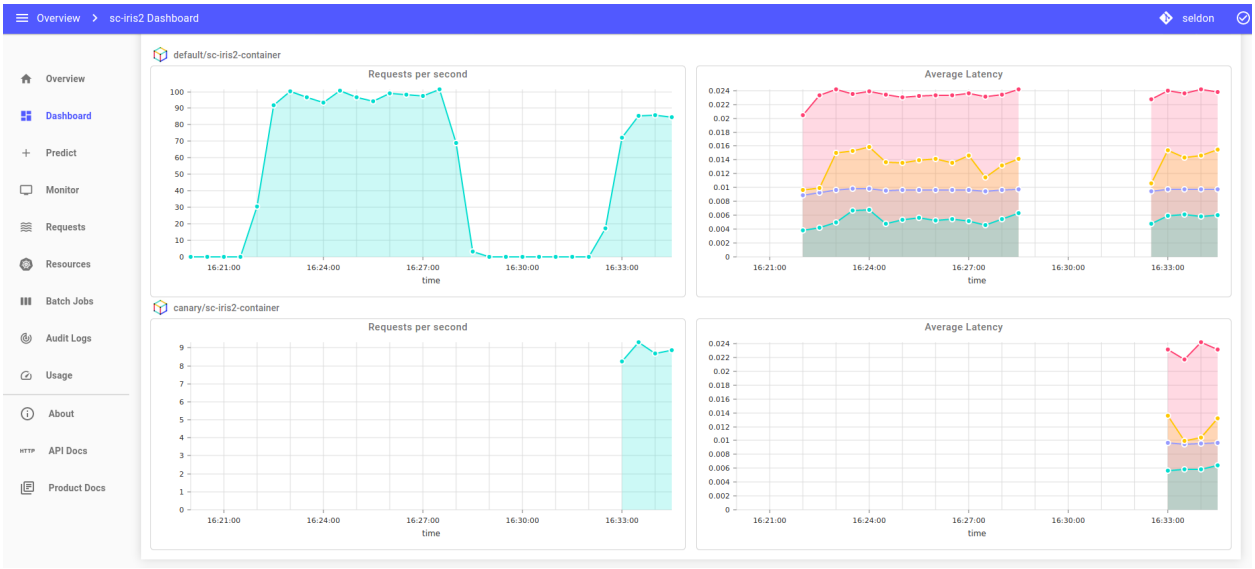
5.1.1.4 Create Canary

Create an XGBoost canary model using the saved model at:

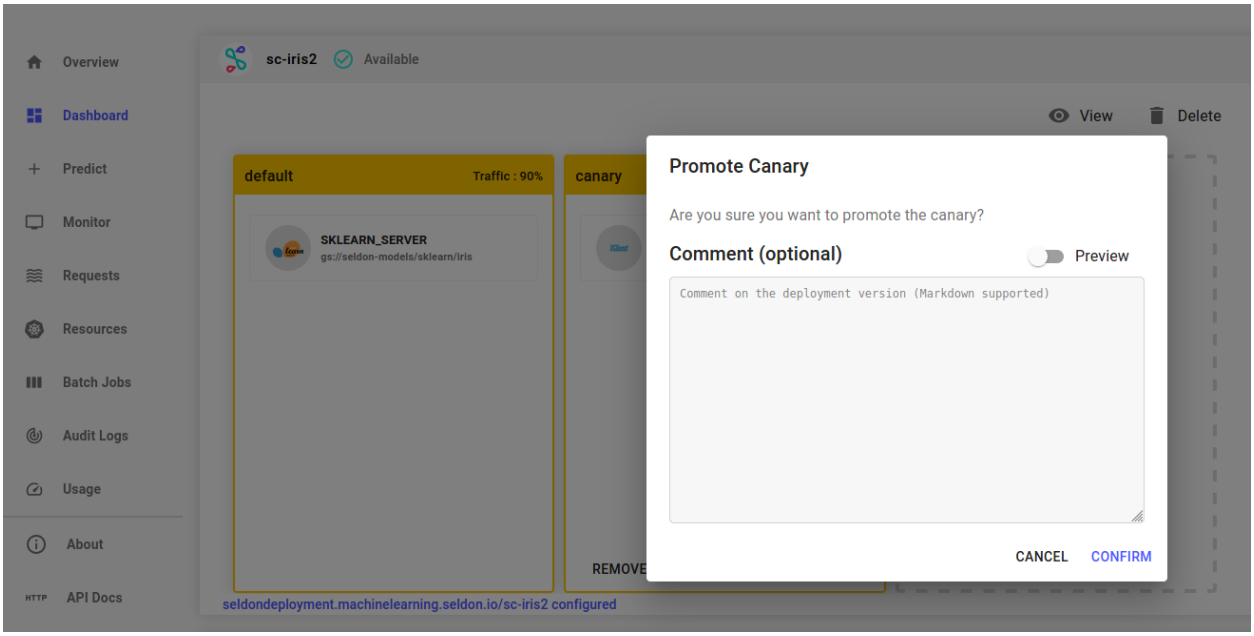
```
gs://seldon-models/xgboost/iris
```



Rerun the load test and you should see metrics for both default and canary models.



Promote the XGBoost Canary to be the main model.



5.1.2 Model Explanations with Anchor Images

In this demo we will:

- Launch an income classification model which has image training features
- Send a request to get a prediction
- Create an explainer for the model
- Send the same request and then get an explanation for it

This model provides a model trained to classify images based on CIFAR10 dataset.

The explainer uses the *anchors technique* to provide insight into why a particular classification was made by the model. We'll see patterns in an input image that are most relevant to the prediction outcome.

5.1.2.1 Create Model

Use the following model uri with tensorflow runtime.

```
gs://seldon-models/tfserving/cifar10/resnet32
```

5.1.2.2 Get Predictions

Run a single prediction using the tensorflow payload format of an image `truck`

5.1.2.3 Add an Anchor Images Explainer

Create a model explainer using the URI below for the saved explainer.

```
gs://seldon-models/tfserving/cifar10/explainer-py36-0.5.2
```

5.1.2.4 Get Explanation for one Request

View all requests and then explain it using the JSON below:

5.1.3 Model Explanations with Anchor Tabular

In this demo we will:

- Launch an income classification model which has tabular training features
- Send a request to get a prediction
- Create an explainer for the model
- Send the same request and then get an explanation for it

This demo uses a model trained to predict high or low income based on demographic features from a 1996 US census.

The explanation will offer insight into why an input was classified as high or low. It uses the *anchors technique* to track features from training data that correlate to category outcomes.

5.1.3.1 Create Model

Use the model uri:

```
gs://seldon-models/sklearn/income/model-0.23.2
```

5.1.3.2 Get Predictions

Run a single prediction using the JSON below.

```
{
  "data": {
    "names": [
      "Age",
      "Workclass",
      "Education",
      "Marital Status",
      "Occupation",
      "Relationship",
      "Race",
      "Sex",
      "Capital Gain",
      "Capital Loss",
      "Hours per week",
      "Country"
    ],
    "ndarray": [
      [
        53,
        4,
        0,
        2,
        8,
        4,
        2,
        0,
        0,
        0,
        60,
        9
      ]
    ]
  }
}
```

5.1.3.3 Add an Anchor Tabular Explainer

Create a model explainer using the URI below for the saved explainer.

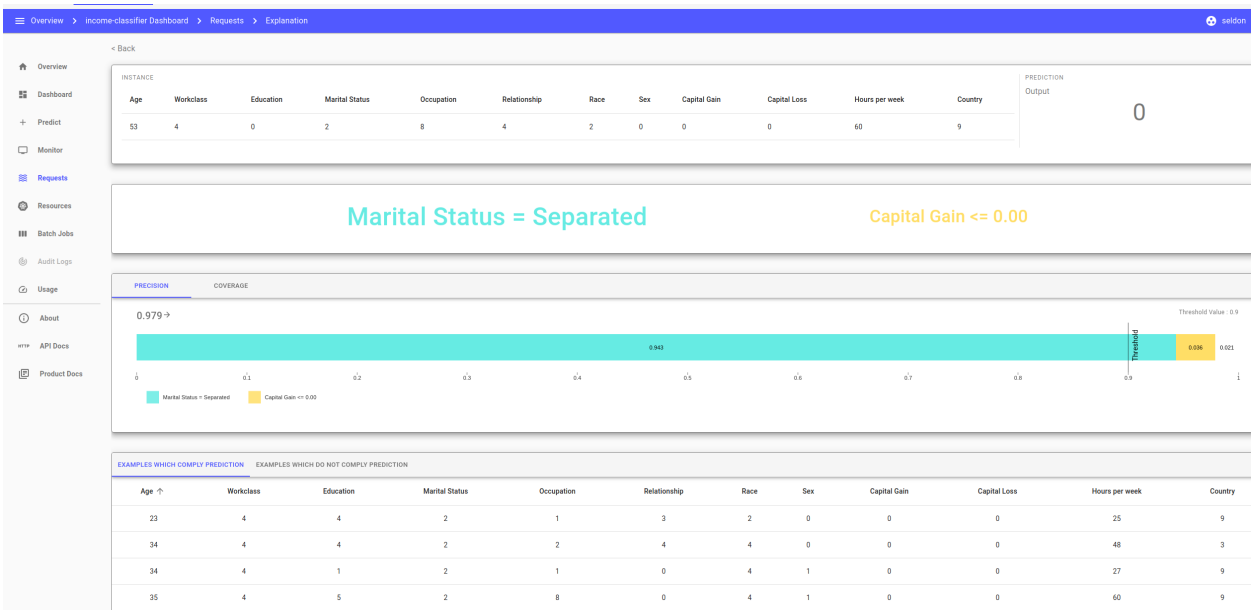
```
gs://seldon-models/sklearn/income/explainer-py36-0.5.2
```

5.1.3.4 Get Explanation for one Request

Resend a single request and then explain it using the JSON below:

```
{
  "data": {
    "names": [
      "Age",
      "Workclass",
      "Education",
      "Marital Status",
      "Occupation",
      "Relationship",
      "Race",
      "Sex",
      "Capital Gain",
      "Capital Loss",
      "Hours per week",
      "Country"
    ],
    "ndarray": [
      [
        53,
        4,
        0,
        2,
        8,
        4,
        2,
        0,
        0,
        0,
        60,
        9
      ]
    ]
  }
}
```

Make the prediction again and go to the 'Request Logs' screen. From there you can click through to the explanation with full feature names.



The anchors reveal which features most contributed to the classification of 'Low Income'. Here, for nearly 98% of the training data a Marital Status of 'Separated' correlates to Low Income. Having no Capital Gain also correlates but is a less strong indicator.

At the bottom we are shown other examples of Low Income data points. This helps to reveal the patterns that the model is relying on to make predictions.

5.1.4 Model Explanations with Anchor Text

In this demo we will:

- Launch a movie sentiment model which takes text input
- Send a request to get a sentiment prediction
- Create an explainer for the model
- Send the same request and then get an explanation for it

The explainer uses the *anchors technique* to provide insight into why a particular classification was made by the model. We'll see patterns in input text that are most relevant to the prediction outcome.

5.1.4.1 Create Model

Use the model uri:

```
gs://seldon-models/sklearn/moviesentiment
```

5.1.4.2 Get Predictions

Run a single prediction using the JSON below.

```
{
  "data": {
    "names": [
      "Text review"
    ],
    "ndarray": [
      "this film has bad actors"
    ]
  }
}
```

5.1.4.3 Add an Anchor Text Explainer

Create an Anchor Text explainer using the default settings.

5.1.4.4 Get Explanation for one Request

Resend a single request using the JSON below and then explain it:

```
{
  "data": {
    "names": [
      "Text review"
    ],
    "ndarray": [
      "this film has bad actors"
    ]
  }
}
```

5.1.5 Model Catalogue

5.1.5.1 Registering Models and Editing Metadata

Notes:

1. Model Catalog is currently experimental.
2. This feature requires *PostgreSQL to be installed*.
3. This demo expects that there is no existing model with the same `URI` or the same combination of `Model Name` and `Version`. The model catalog enforces uniqueness on these fields. Requests that result in a conflict will be rejected.

5.1.5.2 Register New Model

1. From the Deployments Overview page, select `Model Catalog` at the top of the page.
2. Click `Register A New Model` to open the registration modal.
3. Enter the following parameters:

- *Model Name*: `cifar10`
- *URI*: `gs://seldon-models/tfserving/cifar10/resnet32`
- *Artifact Type*: `Tensorflow`
- *Version*: `v1.0`
- *Task Type*: `classification`

1. Append tags and metric parameters with the following values. To add more parameters, use the + button. To delete a parameter, use the `x` button:

- Tags:
  - *key*: `author`, *value*: `Seldon`,
  - *key*: `training_set`, *value*: `Resnet32`,
- Metrics:
  - *key*: `p1`, *value*: `0.8`,
  - *key*: `p2`: *value*: `0.6`

1. Click `Register Model`

5.1.5.3 Edit Model Metadata

From The Model Catalog

1. From the Deployments Overview page, select `Model Catalog` at the top of the page.
2. Select a model. For the purposes of this demonstration, the model registered in the Register New Model step will be used.
3. On the side drawer that opens, click `EDIT METADATA`.
4. Add a new tag with the following values:

- *key*: `stage`, *value*: `production`

1. Click `SAVE METADATA` at the top right hand side of the side drawer to save your edit.

From A Deployment's Dashboard

Note: Please follow the instructions on creating an *Outlier Detector*.

1. From the Deployments Overview page, select the deploment to append metadata on the running model. For the purposes of this demonstration, the deployment created from the *Outlier Detector* example will be used.
2. From your selected deployment's dashboard, click the model being used to open a drawer on the right.
3. Click `EDIT METADATA`.
4. Under the *Tags* section, press the + button to add a new tag.
5. For the new tag, enter the following parameters:

- *key*: `stage`, *value*: `production`

1. Click `SAVE METADATA` on the top right of the side drawer.

Deploying Models From The Model Catalog

Note: This demonstration requires the cifar10 model to have been registered already.

1. From the overview page, navigate to the model catalog using the header tabs.
2. Select the *more* icon for the model named `cifar10`.
3. In the dropdown menu that appears, select *Deploy*.
4. In the wizard which appears, name the *deployment* `cifar10`, and set the protocol to `Tensorflow`. Click `next` when done.
5. On the *Default Predictor Parameters* screen, the model URI should automatically be appended. For Triton models, the *model name* will also be appended automatically. Click *next* to continue.
6. Skip the remaining steps and click `Launch`.

Select `Deployments` header tab to verify that your model has been deployed.

5.1.6 Outlier Detection with CIFAR10 Image Classifier

This demo is based on VAE outlier detection in the alibi detect project. Here we will :

- Launch an image classifier model trianed on the CIFAR10 dataset
- Setup an outlier detector for this particular model
- Send a request to get a image classification
- Send a perturbed request to get a outlier detection

Important: This demo requires Knative installation on the cluster as the outlier detector will be installed as a kservice.

5.1.6.1 Create Model

Use the following model uri with `tensorflow` runtime. Set the protocol to 'tensorflow':

```
gs://seldon-models/tfserving/cifar10/resnet32
```

5.1.6.2 Setup Outlier detector

Setup an outlier detector with model name `cifar10` using the default settings (which sets Reply URL as seldon-request-logger in the logger's default namespace - change if you modified this at install time) and storage URI as follows:

```
gs://seldon-models/alibi-detect/od/OutlierVAE/cifar10
```

5.1.6.3 Make Predictions

Run a single prediction using the tensorflow payload format of an image `truck`. Also a perturbed image of the truck in the same format at `outlier truck image`. Make a couple of these requests at random using the predict tool in the UI.

5.1.6.4 View outliers on the Requests Screen

Go to the requests screen to view all the historical requests. You can see the outlier value on each instance. Also you can highlight outliers based on this score and also use the filter to see only the outliers as needed.

5.1.6.5 Monitor outliers on the Monitor Screen

Under the 'Monitor' section you can see a timeline of outlier requests.

5.1.6.6 Troubleshooting

If you experience issues with this demo, see the *troubleshooting docs* and also the *knative* or *elasticsearch* sections.

5.1.7 Drift Detection with CIFAR10 Image Classifier

This demo is based on model distiliation drift detector in the alibi detect project.

The idea is that input data can change over time and become significantly different from the model training data. When the input data distribution shifts then prediction quality can drop. We want to know when that happens.

Here we will :

- Launch an image classifier model trianed on the CIFAR10 dataset.
- Setup a drift detector for this particular model.
- Send a request to get a image classification.
- Send a request to trigger a drift detection.
- View the drift in the dashboard.

Important: This demo requires Knative installation on the cluster as the outlier detector will be installed as a kservice.

5.1.7.1 Create A Model

Note

1. Verify the namespace you will be deploying to.
2. Deployment names must be unique and only contain alphanumeric characters.

From the *deployment overview* screen, click on the Create button to create a new deployment.

1. In the deployment creation wizard, enter a name for your new deployment.
2. Select the namespace you would like the deployment to reside in *(e.g.* `seldon`*)*.
3. From the *protocol* dropdown menu, select `Tensorflow` and click next.
4. For the deployment details, enter the following values, then click next:
   - Runtime: `Tensorflow`
   - Model URI:

   ```
   gs://seldon-models/tfserving/cifar10/resnet32
   ```

5. Skip the remaining steps, then click `Launch`.

5.1.7.2 Add A Drift Detector

Note

1. Make sure you are in the correct namespace.
2. By default, the Reply URL is set as `seldon-request-logger` in the logger's *default namespace*. **If you are using a custom installation, please change this parameter according to your installation**.
3. Increase this value to make the drift detector less sensitive.

From the *deployment overview page*, select your `cifar10` deployment to enter the deployment dashboard.

Inside the *deployment dashboard*, add a drift detector with by clicking the `Create` button within the `Drift Detection` widget.

Enter the following parameters in the modal popup which appears, using the default settings:

- *Model Name*: `cifar10`.
- *Model URI*:

```
gs://seldon-models/alibi-detect/cd/ks/cifar10-0_4_4
```

- *Reply URL*:

```
http://seldon-request-logger.seldon-logs
```

- *Batch Size*: `2`.
- *Protocol*: `Tensorflow`.
- *HTTP Port*: `8080`.

Then, click `Create Drift-Detector` to complete the setup.

5.1.7.3 Make Predictions

1. From the deployment dashboard, click on `make new prediction`.
2. Run a single prediction using the tensorflow payload format of an image `truck`

Next, run a single prediction using the tensorflow payload format of an image of an `outlier truck image`.

5.1.7.4 Monitor Drift On The Monitor Screen

Under the 'Monitor' section of your deployment, you can see a timeline of drift requests.

This becomes clearer if you make many requests over a period of time.

5.1.7.5 Troubleshooting

If you experience issues with this demo, see the *troubleshooting docs* and also the *knative* or *elasticsearch* sections.

5.1.8 Model Accuracy Metrics with Iris Classifier

Iris is the family in the flower which contains the several species such as the setosa, versicolor, virginica,etc. This demo is based on Iris classification model based on flower properties like Sepal length, Sepal width, Petal length, Petal width. Here we will :

- Launch an iris classifier model
- Setup an metrics server for this particular model
- Send a request to get a iris classification
- Send feedback requests to get a gather accuracy metrics

Important: This demo requires Knative installation on the cluster as the metrics server will be installed as a kservice. Also the metrics server only works with classification models in this version.

5.1.8.1 Create Model

Use the following model uri with `seldon` runtime. Set the protocol to 'seldon':

```
gs://seldon-models/sklearn/iris
```

5.1.8.2 Setup Metrics Server

Setup an metrics server with model name `multiclassserver` using the default settings (which sets Reply URL as seldon-request-logger in the logger's default namespace - change if you modified this at install time) and storage URI as follows:

```
adserver.cm_models.multiclass_numeric.MultiClassNumeric
```

And set the protocol as `Seldon Feedback` for port `8080`.

5.1.8.3 Make Predictions

Run a single prediction using the ndarray payload format. Make a couple of these requests at random using the predict tool in the UI.

```
{
        "data": {
                "names": ["Sepal length", "Sepal width", "Petal length", "Petal Width
↪"],
                "ndarray": [
                        [6.8, 2.8, 4.8, 1.4]
                ]
        }
}
```

The prediction response is `versicolor`.

```
{
        "data": {
                "names": [
                        "t:0",
                        "t:1",
                        "t:2"
                ],
                "ndarray": [
                        [
                                0.008074020139120223,
                                0.7781601484223125,
                                0.21376583143856714
                        ]
                ]
        },
        "meta": {}
}
```


5.1.8.4 Send Feedback

As we saw the prediction response was `versicolor`. In numeric form the response is,


```
{
  "data": {
    "ndarray": [
      1
    ]
  }
}
```


Now prepare a feedback with the prediction response and the truth for the numeric metrics server as follows:


```
{
        "response": {
                "data": {
                        "ndarray": [
                                1
                        ]
                }
        },
        "truth": {
                "data": {
                        "ndarray": [
                                1
                        ]
                }
        }
}
```


Now send a feedback request as following with the auth token from the curl form in the predict tool. This feedback represents a case of true positive.

```
CLUSTER_IP=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}')
curl -k -H "X-Auth-Token: $AUTH_TOKEN" -H "Content-Type: application/json" https://
↪$CLUSTER_IP/seldon/seldon/iris-classifier/api/v0.1/feedback -d '{"response":{"data":
↪{"ndarray":[1]}},"truth":{"data":{"ndarray":[1]}}}'
```

Also prepare a feedback with the prediction response and the truth for the numeric metrics server such that the truth is different from the prediction.

```
{
        "response": {
                "data": {
                        "ndarray": [
                                1
                        ]
                }
        },
        "truth": {
                "data": {
                        "ndarray": [
                                0
                        ]
                }
        }
}
```

Now send a feedback request as following with the auth token from the curl form in the predict tool. This feedback represents a case of false positives and negetives.

```
CLUSTER_IP=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}')
curl -k -H "X-Auth-Token: $AUTH_TOKEN" -H "Content-Type: application/json" https://
↪$CLUSTER_IP/seldon/seldon/iris-classifier/api/v0.1/feedback -d '{"response":{"data":
↪{"ndarray":[1]}},"truth":{"data":{"ndarray":[0]}}}'
```

5.1.8.5 Monitor accuracy metrics on the Monitor Screen

Go to the monitor screen's the accuracy metrics tab to view all the metrics. Set the time range to view the metrics. You can see metrics like accuracy, precision, recall and sepcificity here. Notice the drop in accuracy metrics after the false feedback was received.

5.1.8.6 Submit batch feedback using Batch Processor component

Now we will submit a feedback as a batch using the Batch Processor component.

We will use two files, each containing 10k feedback instances:

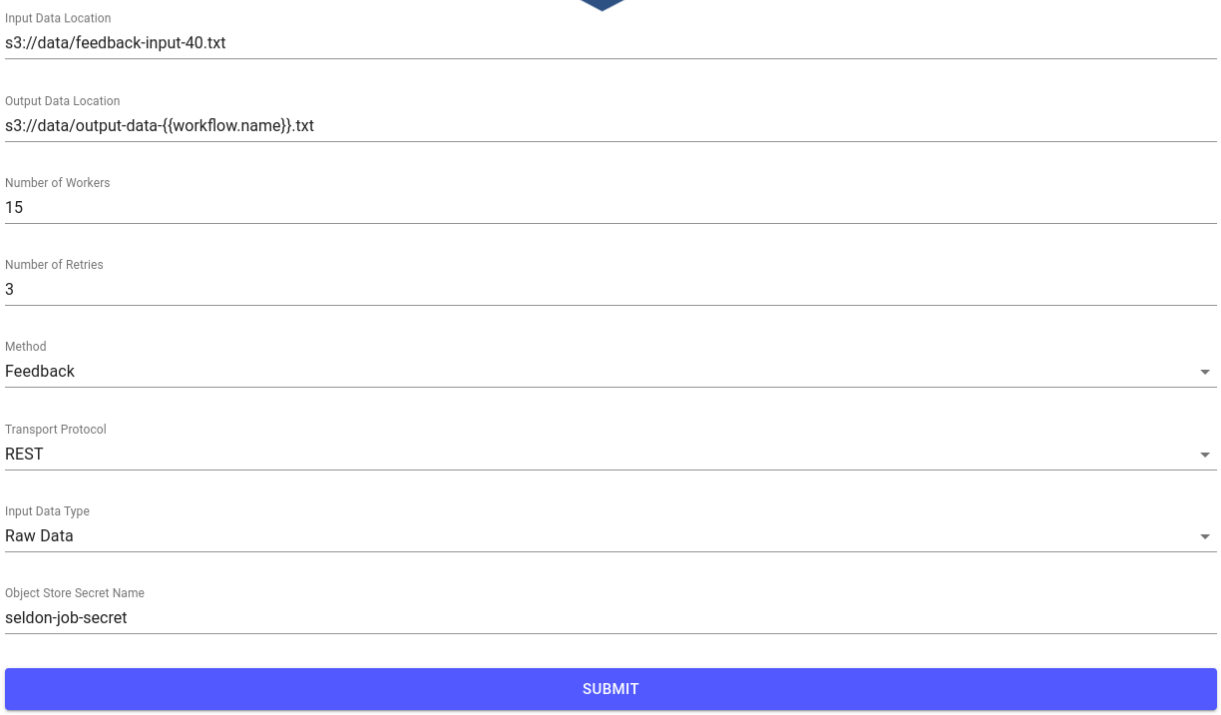
- `90% success rate feedback file`
- `40% success rate feedback file`

Ww need to upload the files to MinIO's `data` bucket. For details on interacting with MinIO UI please see *Batch Demo*.

Once files are in S3 bucket, we go to `Batch Requests` screen using following button available at the main model screen

and submit batch request using following form values for both `feedback-input-90.txt` and `feedback-input-40.txt` files

```
Input Data Location: s3://data/feedback-input-40.txt
Output Data Location: s3://data/output-data-{{workflow.name}}.txt
Number of Workers: 15
Number of Retries: 3
Method: Feedback
Transport Protocol: REST
Input Data Type: Raw Data
Object Store Secret Name: seldon-job-secret
```



Now go to the monitor view and observe how metrics value evolve over time.

5.1.8.7 Troubleshooting

If you experience issues with this demo, see the *troubleshooting docs* and also the *knative* or *elasticsearch* sections.

5.1.9 Batch Requests

5.1.9.1 Pre-requisites

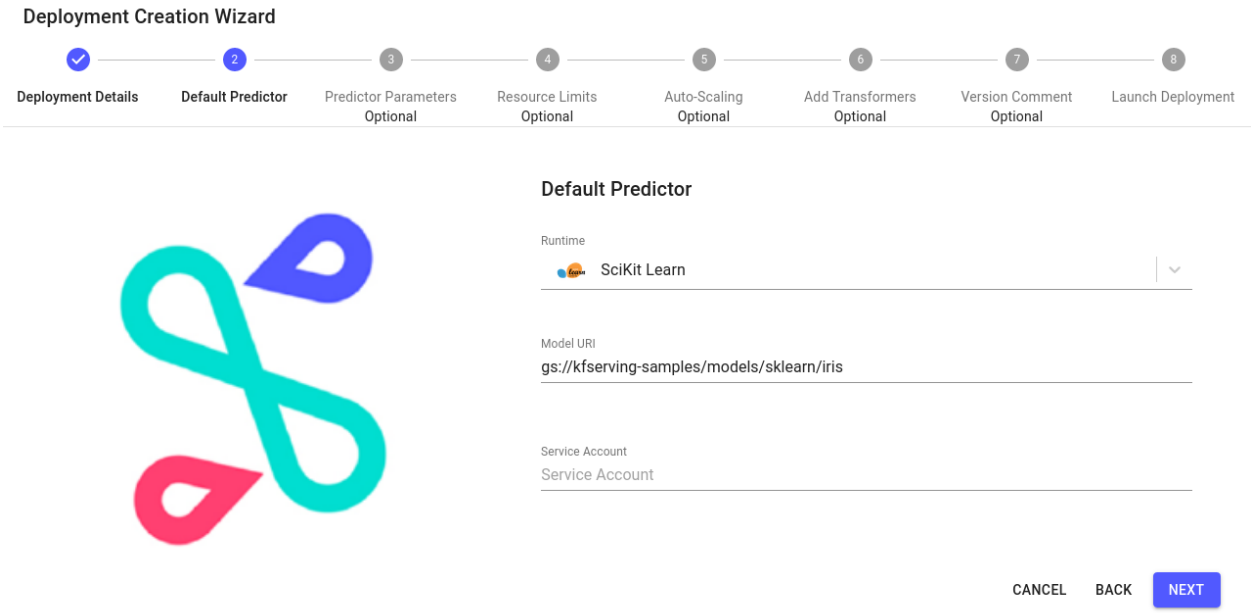
Minio should already be installed with Seldon Deploy. The minio browser should be exposed on `/minio/`. For a managed trial cluster by default the credentials will be as per the deploy login.

On a production cluster the namespace needs to have been setup with a service account. This can be found under the *argo install documentation*.

We will:

- Deploy a pretrained sklearn iris model
- Run a batch job to get predictions
- Check the output

5.1.9.2 Deploy Model



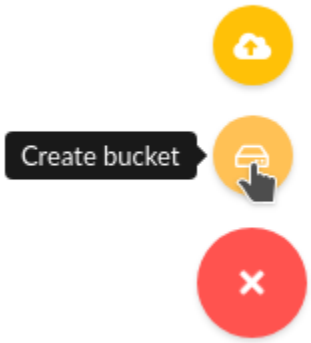
Use the model uri:

```
gs://seldon-models/sklearn/iris
```

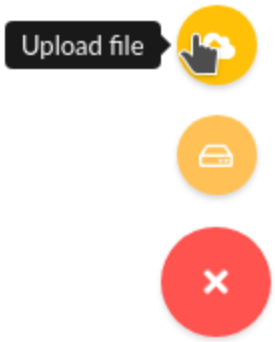
5.1.9.3 Setup Input Data

Download the `input data file`.

Go to the minio browser and use the button in the bottom-right to create a bucket. Call it `data`.



Again from the bottom-right choose to upload the `input-data.txt` file to the `data` bucket.



5.1.9.4 Run a Batch Job

Go to 'Batch Jobs' in the middle-right under the text 'Initiate or get the status of batch requests'. Create a batch job with parameters:

```
Input Data Location: s3://data/input-data.txt
Output Data Location: s3://data/output-data-{{workflow.name}}.txt
Number of Workers: 15
Number of Retries: 3
Method: Predict
Transport Protocol: REST
Input Data Type: ndarray
Object Store Secret Name: seldon-job-secret
```

Input Data Location
s3://data/input-data.txt

Output Data Location
s3://data/output-data-{{workflow.name}}.txt

Number of Workers
15

Number of Retries
3

Method
Predict

Transport Protocol
REST

Input Data Type
ndarray

Object Store Secret Name
seldon-job-secret

SUBMIT

Here `seldon-job-secret` is a pre-created secret *in the namespace containing env vars*.

Give the job 1-2 minutes to complete. Then refresh the page to see the status.

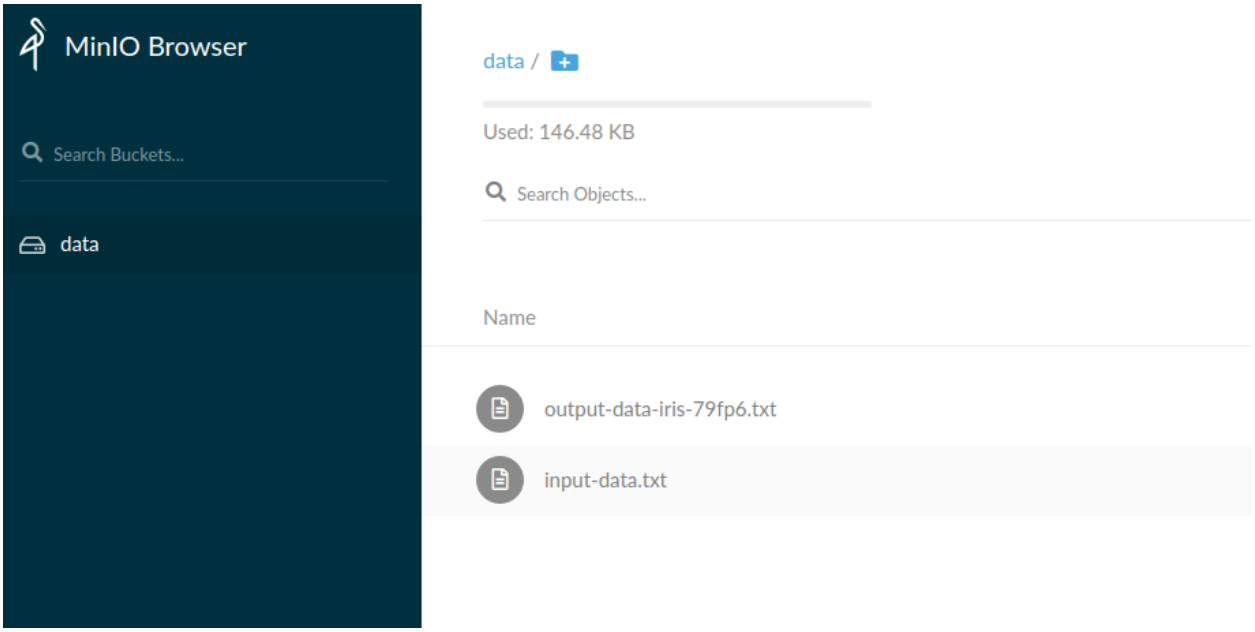
Latest Job

iris-kbgz5

C38C98A7-5468-46B2-B997-F8541FEF9281

started at:2020-11-23T17:03:01Z
finished at:
Status:Running

In minio you should now see an output file:



If you open that file you should see contents such as:


```
{"data": {"names": ["t:0", "t:1", "t:2"], "ndarray": [[0.0006985194531162841, 0.
↪003668039039435755, 0.9956334415074478]]}, "meta": {"requestPath": {"iris-container
↪": "seldonio/sklearnserver:1.5.0-dev"}, "tags": {"tags": {"batch_id": "8a8f5e26-
↪2b44-11eb-8723-ae3ff26c8be6", "batch_index": 3.0, "batch_instance_id": "8a8ff94e-
↪2b44-11eb-b8d0-ae3ff26c8be6"}}}}
{"data": {"names": ["t:0", "t:1", "t:2"], "ndarray": [[0.0006985194531162841, 0.
↪003668039039435755, 0.9956334415074478]]}, "meta": {"requestPath": {"iris-container
↪": "seldonio/sklearnserver:1.5.0-dev"}, "tags": {"tags": {"batch_id": "8a8f5e26-
↪2b44-11eb-8723-ae3ff26c8be6", "batch_index": 6.0, "batch_instance_id": "8a903666-
↪2b44-11eb-b8d0-ae3ff26c8be6"}}}}
{"data": {"names": ["t:0", "t:1", "t:2"], "ndarray": [[0.0006985194531162841, 0.
↪003668039039435755, 0.9956334415074478]]}, "meta": {"requestPath": {"iris-container
↪": "seldonio/sklearnserver:1.5.0-dev"}, "tags": {"tags": {"batch_id": "8a8f5e26-
↪2b44-11eb-8723-ae3ff26c8be6", "batch_index": 1.0, "batch_instance_id": "8a8fbe98-
↪2b44-11eb-b8d0-ae3ff26c8be6"}}}}
```

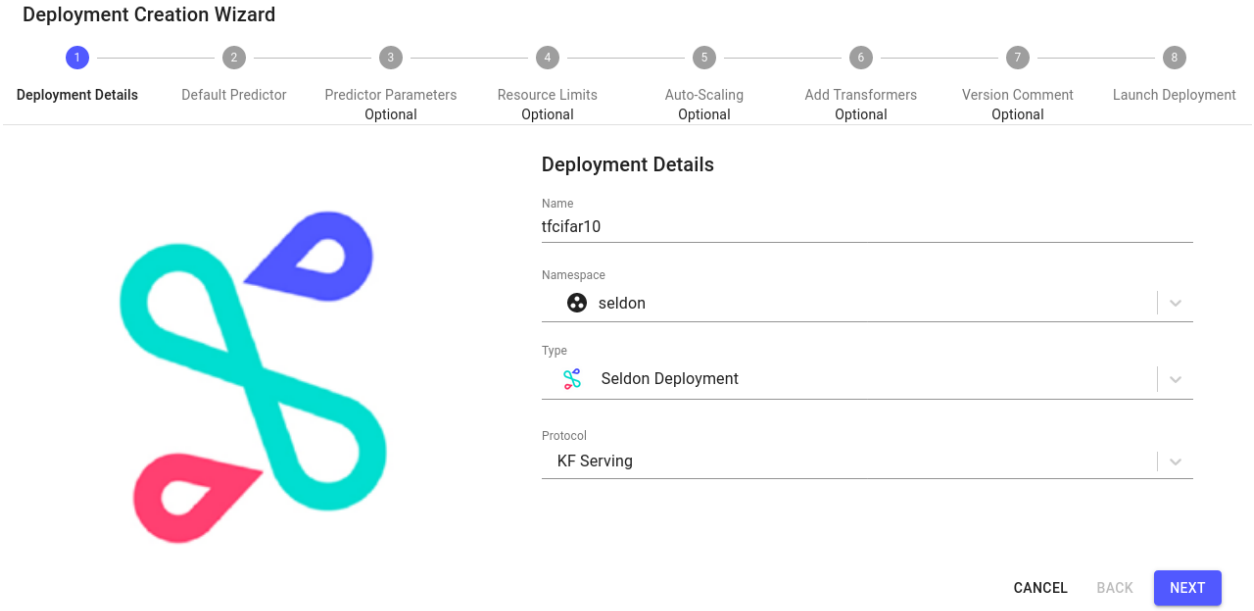

If not, see the *argo section* for troubleshooting.

5.1.10 NVIDIA Triton Server and Alibi Explanations

In this demo we will deploy an image classification model on NVIDIA Triton with GPUs and run explanations using Seldon Alibi. This demo also uses the KFserving V2 protocol for model prediction and explanation payload. Learn more about V2 protocol at Predict Protocol - Version 2 git repository.

5.1.10.1 Deploy an image classifier model

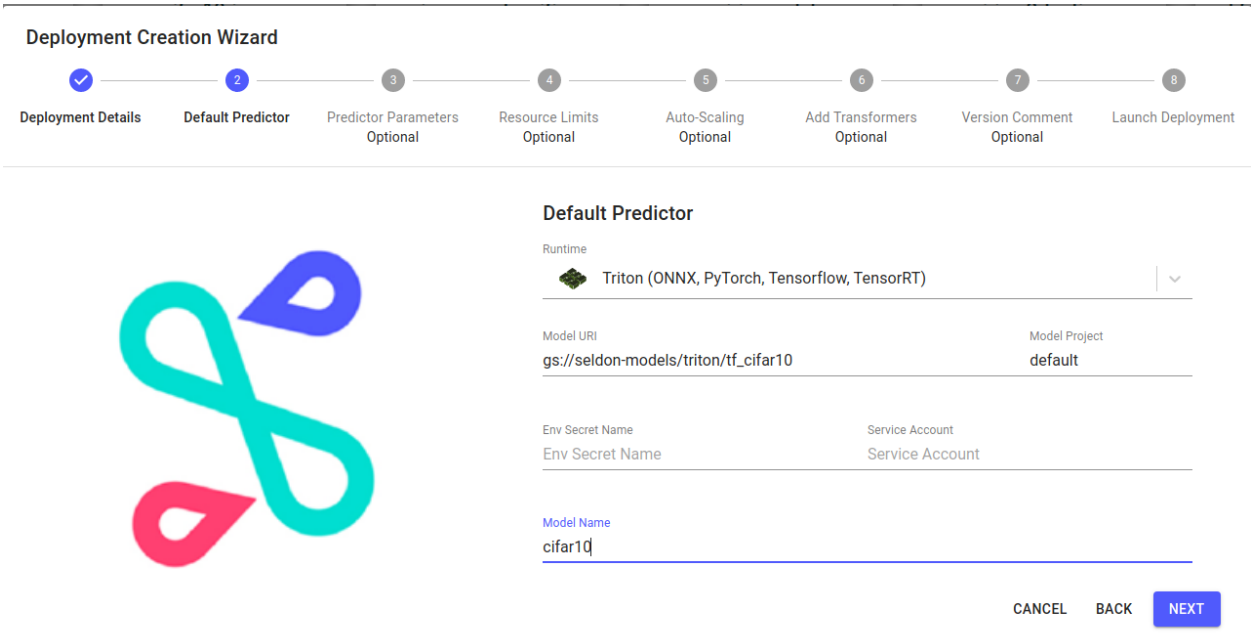
For this example choose `tfcifar10` as the name and use the KFServing protocol option.



For the model to run we have created several image classification models from the CIFAR10 dataset.

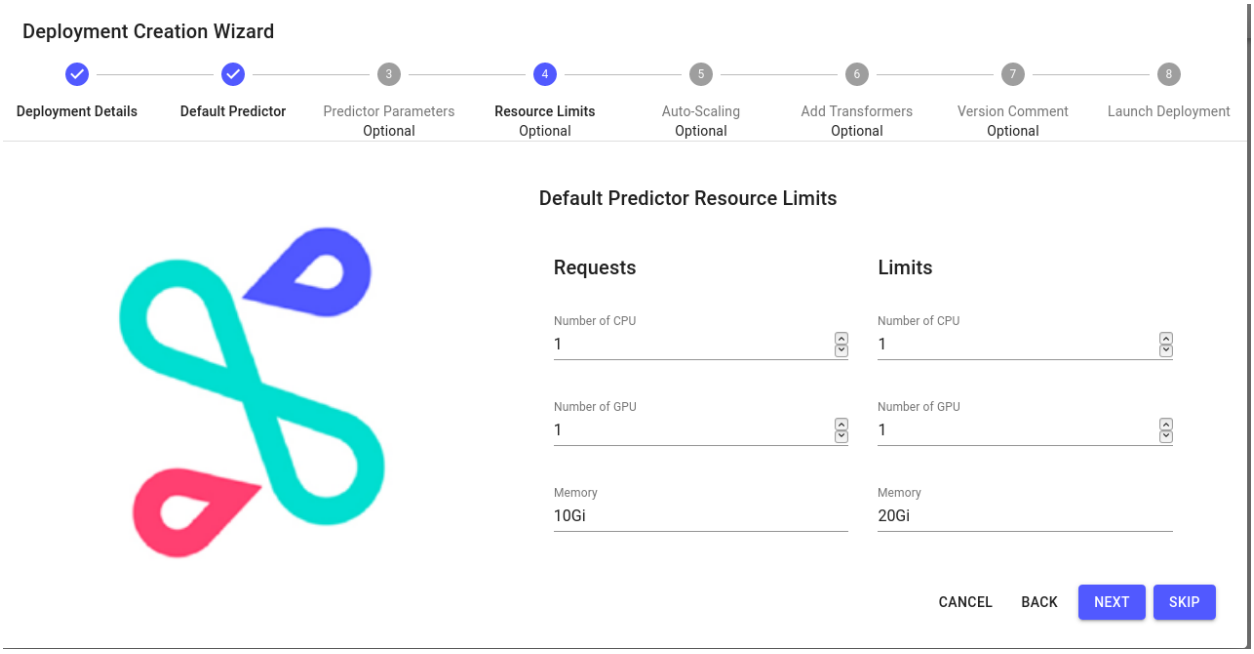
- Tensorflow Resnet32 model: `gs://seldon-models/triton/tf_cifar10`
- ONNX model: `gs://seldon-models/triton/onnx_cifar10`
- PyTorch Torchscript model: `gs://seldon-models/triton/pytorch_cifar10`

Choose one of these and select Triton as the server. Customize the model name to that of the name of the model saved in the bucket for Triton to load.



Configure NVIDIA GPU resources

Next, on the resources screen add 1 GPU request/limit assuming you have these available on your cluster and ensure your have provided enough memory for the model. To determine these settings we recommend you use the NVIDIA model analyzer.



5.1.10.2 Make model predictions

When ready you can test with images. The payload will depend on the model from above you launched.

- `Resnet32`
- `ONNX`
- `PyTorch`

5.1.10.3 Configure an Alibi Anchor Images Explainer

The explanation will offer insight into why an input was classified as high or low. It uses the *anchors technique* to track features from training data that correlate to category outcomes. Create a model explainer using the URI below for the saved explainer.

```
gs://seldon-models/tfserving/cifar10/explainer-py36-0.5.2
```

5.1.10.4 Get Explanation for a single prediction

View all requests and then click the alibi icon to run an explanation request. Note that the explanation request is also made as the same KFserving V2 protocol payload.

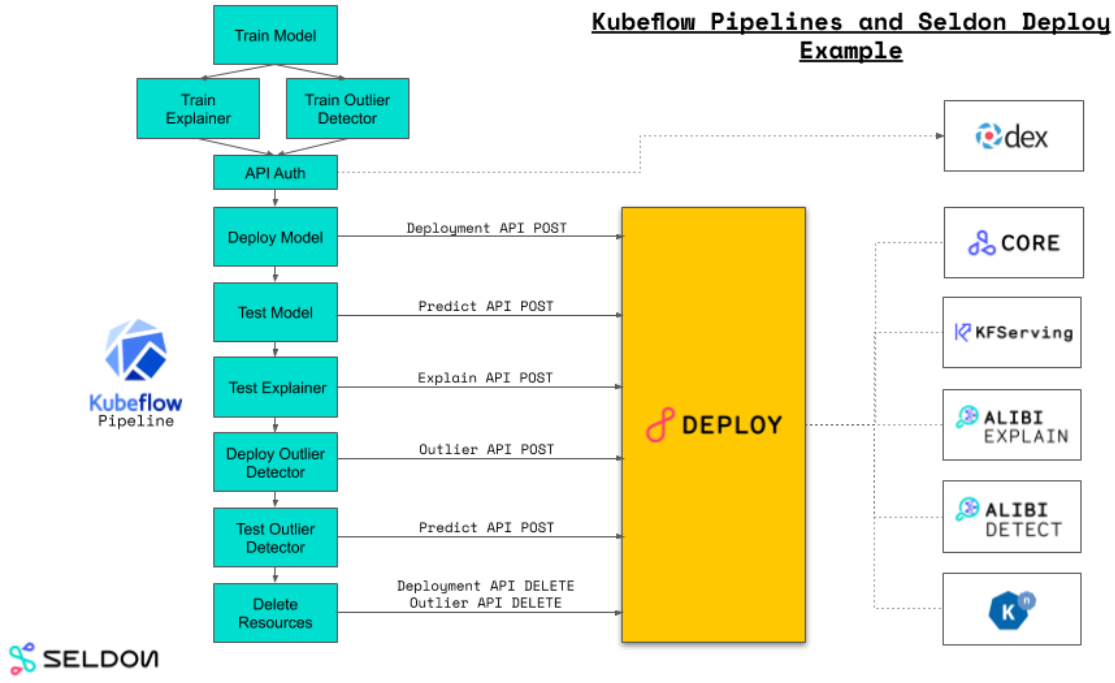
5.1.11 Kubeflow Example

5.1.11.1 Kubeflow Pipeline with Kale example using the Seldon Deploy Enterprise API

We use demographic features from the 1996 US census to build an end to end machine learning pipeline. The pipeline is also annotated so it can be run as a Kubeflow Pipeline using the Kale pipeline generator.

The notebook/pipeline stages are:

1. Setup
    - Imports
    - pipeline-parameters
    - minio client test
2. Train a simple sklearn model and push to minio
3. Prepare an Anchors explainer for model and push to minio
4. Test Explainer
5. Train an isolation forest outlier detector for model and push to minio
6. Deploy a Seldon model and test using the Seldon Deploy Enterprise API
7. Deploy an outlier detector with the Seldon Deploy Enterprise API
8. Test the outlier detector



5.1.11.2 Prerequisites

- A Seldon Deploy install, version >=1.0.0, installed with Kubeflow.
- Obtain the code for this demo by running from the seldon-deploy-resources Github repository.

```
git clone https://github.com/SeldonIO/seldon-deploy-resources.git
cd examples/seldon-core/kubeflow
```

5.1.11.3 GCP Setup

For a GCP cluster we need a RWX Persistent Volume for the shared data Kale needs. To set this up on GCP update and run the script `create-pv.sh` after setting the values for your project, Filestore name and Zone:

```
PROJECT=seldon-demos
FS=pipeline-data
ZONE=europe-west1-b

gcloud beta filestore instances create ${FS}     --project=${PROJECT}     --zone=$
↪{ZONE}     --tier=STANDARD     --file-share=name="volumes",capacity=1TB     --
↪network=name="default",reserved-ip-range="10.0.0.0/29"

FSADDR=$(gcloud beta filestore instances describe ${FS} --project=${PROJECT} --zone=$
↪{ZONE} --format="value(networks.ipAddresses[0])")

helm install nfs-cp stable/nfs-client-provisioner --set nfs.server=${FSADDR} --set␣
↪nfs.path=/volumes --namespace=kubeflow

kubectl rollout status  deploy/nfs-cp-nfs-client-provisioner -n kubeflow
```

The project and zone should match your GCP kubernetes cluster. FS name can be `pipeline-data`.

If you build the pipeline Python DSL using Kale from the notebook you will at present need to modify the created pyhton and change the Kale `VolumeOp` by adding a `storage_class` for the NFS PV, for example:

```
marshal_vop = dsl.VolumeOp(
     name="kale-marshal-volume",
     resource_name="kale-marshal-pvc",
     storage_class="nfs-client",
     modes=dsl.VOLUME_MODE_RWM,
     size="1Gi")
```

5.1.11.4 RBAC Setup

The default pipeline-runner service account needs to be modified to allow creation of secrets and knative triggers.

As an admin user run:

```
kubectl create -f pipeline-runner-additions.yaml
```

5.1.11.5 Pipeline/Notebook Parameters

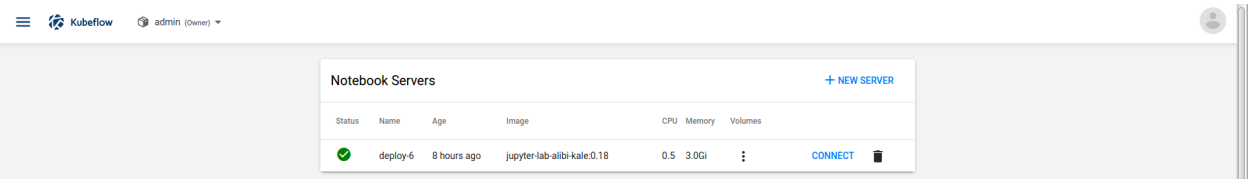
The pipeline/notebook has several core parameters that will need to be set correctly.

Name	Default Value
DEPLOY_NAMESPACE	admin
DEPLOY_PASSWORD	12341234
DEPLOY_SERVER	https://x.x.x.x.x/seldon-deploy/
DEPLOY_USER	admin@kubeflow.org
EXPLAINER_MODEL_PATH	sklearn/income/explainer
INCOME_MODEL_PATH	sklearn/income/model
MINIO_ACCESS_KEY	admin@seldon.io
MINIO_HOST	minio-service.kubeflow:9000
MINIO_MODEL_BUCKET	seldon
MINIO_SECRET_KEY	12341234
OUTLIER_MODEL_PATH	sklearn/income/outlier

If you're not sure of your minio credentials, run `kubectl get secret -n kubeflow mlpipeline-minio-artifact -o yaml` and base64 decode them.

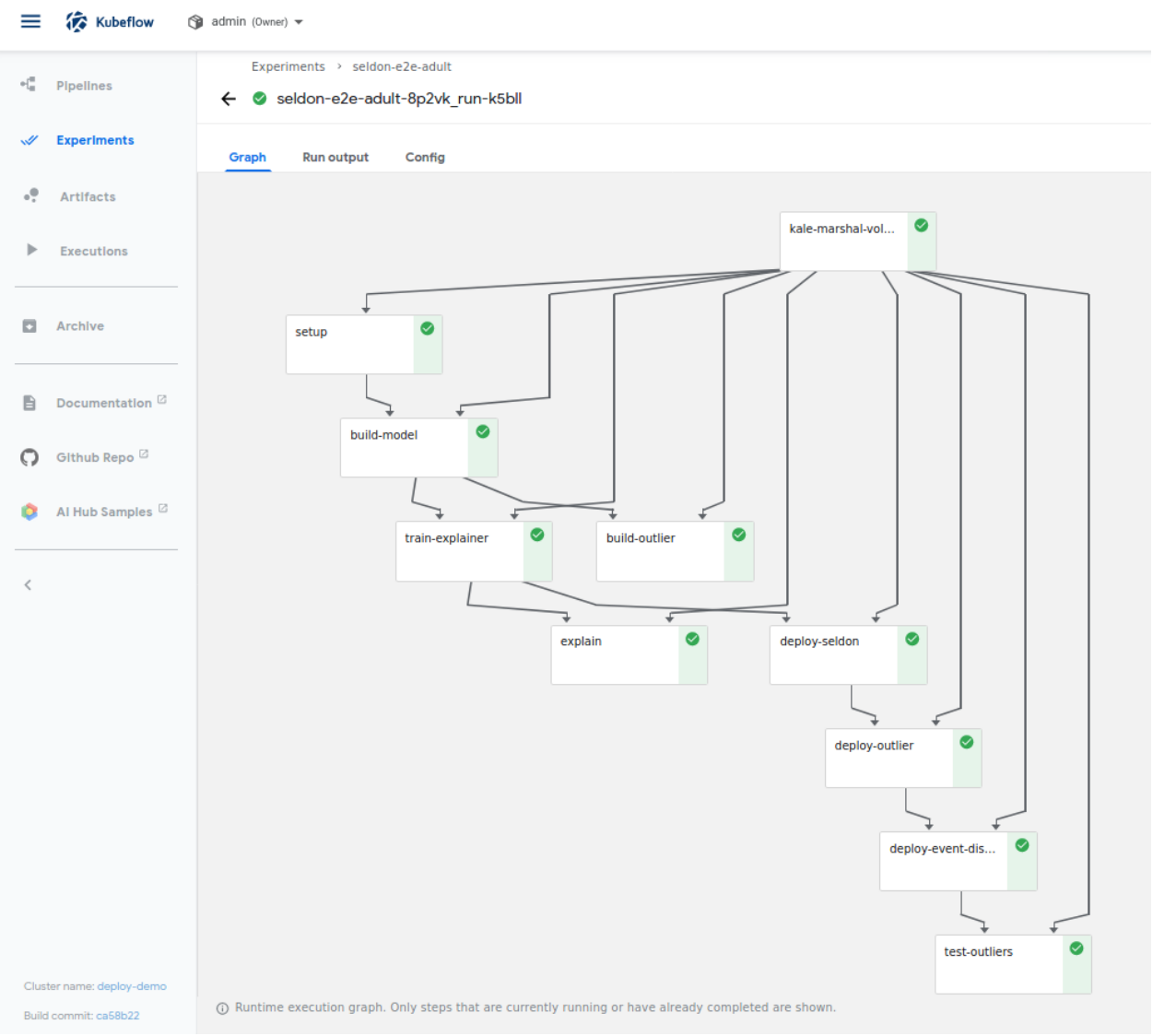
5.1.11.6 Kubeflow Notebook Server

Start a Kubeflow Notebook server with custom image `seldonio/jupyter-lab-alibi-kale:0.23` (other settings default)



5.1.11.7 Test Pipeline

Assuming you have run the GCP and RBAC setup above you can launch the pipeline saved in `seldon_e2e_adult_nfs.kale.py`.



The yaml file can be uploaded to the pipelines UI and then create an experiment and a run.

5.1.11.8 Tested on

If you have tested the pipeline successfully please add a PR to extend the table below.

K8S	Kube-flow	Knative Eventing	Sel-don	KF-Serving	Kale	Notes
GKE 1.16.13	1.2	0.18	1.5.0	0.4.0	0.5.0	GCP Setup above, Kale storage_class fix, RBAC update above
GKE 1.16.15	1.2	0.18	1.5.0	0.4.1	0.5.0	RBAC update and disable istio sidecars

5.1.12 Distributions Monitoring

Distributions monitoring provides an ability to view the statistics and distributions of features and predictions made by your model between any given time. This feature also enables you to draw comparisons between the model predictions for different feature combinations, cohorts and/or time slices. It is a vital aspect of model monitoring cycle to understand if the deployed model has the desired prediction characteristics during different times and for different cohorts.

This demo uses a model trained to predict high or low income based on demographic features from a 1996 US census. In this demo we will observe the predictions and feature distributions of live predictions made using this model with the following steps:

- Register an income classifier model with the relevant predictions schema
- Launch a Seldon Core deployment with the income classifier model
- Make predictions using a REST requests to the model deployment
- Observe the feature distributions of the live predictions
- Filter distributions by time or predictions and feature level filters

---

Note: This demo needs the request logger to connect to Seldon Deploy in order to fetch model level predictions schema. And this requires specific request logger configuration. Also this feature is supported with many protocols available with deployments like seldon, tensorflow and the kfserving v2 protocol. But not supported for json data, string data, bytes payload or multi-node graph use cases yet.

5.1.12.1 Register a income classifer model

Register the income classifier SKLearn model with the below URI.

```
gs://seldon-models/sklearn/income/model-0.23.2
```

5.1.12.2 Configure predictions schema

Edit the model metadata to update the predictions schema for the model. The predictions schema is a generic schema structure for machine learning model predictions. It is a definition of feature inputs and output targets from the model prediction. Use the `income classifier model predictions schema` to edit and save the model level metadata. Learn more about the predictions schema at the ML Predictions Schema open source repository.

5.1.12.3 Launch a Seldon Core deployment

Deploy the income classifier model from the catalogue into an appropriate namespace.

5.1.12.4 Make predictions using the model deployment

Model predictions can be made in the appropriate protocol. In this demo, we use seldon protocol, see a single prediction payload example below,

```
{
  "data": {
    "names": [
      "Age",
      "Workclass",
      "Education",
      "Marital Status",
      "Occupation",
      "Relationship",
      "Race",
      "Sex",
      "Capital Gain",
      "Capital Loss",
      "Hours per week",
      "Country"
    ],
    "ndarray": [[53, 4, 0, 2, 8, 4, 2, 0, 0, 0, 60, 9]]
  }
}
```

Distributions monitoring is especially useful to keep track of predictions when a model makes thousands of predictions in real world scenario. To simulate such a use case, make multiple predictions over time in the Seldon protocol request format using the `predictions data csv file` and the following shell script which makes around `32560 predictions with an interval of 5 seconds between each request`. Note that you need sufficient payload logging infra needed for this.

```
CLUSTER_IP=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}')
DEPLOYMENT_NAME=income-classifier
DEPLOYMENT_NAMESPACE=seldon
INTERVAL_SECONDS=5

while IFS= read -r line; do
    curl -k -H "Content-Type: application/json" \
        http://$CLUSTER_IP/seldon/$DEPLOYMENT_NAMESPACE/$DEPLOYMENT_NAME/api/v0.1/
↪predictions \
        -d '{"data":{"names":["Age","Workclass","Education","Marital Status",
↪"Occupation","Relationship","Race","Sex","Capital Gain","Capital Loss","Hours per␣
↪week","Country"],"ndarray":[['${line// /,}']]}}'
    sleep "${INTERVAL_SECONDS}s"
done <prediction-data.csv
```

5.1.12.5 Observe predictions and feature distributions

Select the income classifier deployment and go to the monitor section to view the predictions and feature distributions.

5.1.12.6 Filter distributions by time or feature level filters

Filter distributions by time or predictions and feature level filters to compare different cohorts and further analysis. For example let's look at the predictions for all individuals in the `Age` group 25-50 and also filter by their `Education` as High-School Grads and Dropouts Only and see how the average prediction frequency changes for this cohort.

5.1.12.7 Configuring parameters

Distributions parameters configuration allows you to configure your charts for further analysis. For example let's look at at the charts in the `Age` group and change the `Histogram interval` to `11` and `Number of time buckets` to `30` to see.

5.1.13 Project based authorization

Notes:

1. Project based authorization is currently experimental.
2. This feature requires *PostgreSQL to be installed*, and *OPA authorization to be enabled*.

5.1.13.1 Setup

1. Make sure Open Policy Agent authorization is enabled as per the *installation guide*.
2. Set the policies in the `seldon-deploy-policy` config map as shown here:

```
cat <<EOF | kubectl apply -n seldon-system -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: seldon-deploy-policies
data:
  data: '{
  "role_grants": {
    "data-scientist": [
      {
        "resource": "project/iris",
        "action": "read"
      }
    ]
  },
  "user_grants": {
    "*": [
      {
        "resource": "namespace/seldon",
```

(continues on next page)

(continued from previous page)

```
        "action": "read"
      },
      {
        "resource": "namespace/seldon",
        "action": "write"
      },
      {
        "resource": "project/default",
        "action": "read"
      },
      {
        "resource": "project/default",
        "action": "write"
      }
    ],
    "alice": [
      {
        "resource": "project/income",
        "action": "write"
      },
      {
        "resource": "project/income",
        "action": "read"
      },
      {
        "resource": "project/iris",
        "action": "read"
      },
      {
        "resource": "project/iris",
        "action": "write"
      }
    ]
  }
}'
EOF
```

Note how all users (`*`) will have access to the `seldon` namespace and `default` project. Seldon Deploy should automatically pick up the changes in the config map after a few seconds, but if you want to reload it immediately you can restart the Seldon Deploy pod.

1. Create two users. If using the default *installation with Keycloak* you can create a user as described in the official documentation and you can assign them to a group as described there as well. Note you might have to create the `data-scientist` group first. The two users must be:
    1. `alice` with no groups associated with her. According to the policies above, she will have read/write access to 3 projects - `default`, `iris`, and `income`.
    2. `bob` in the `data-scientist` group. According to the policies above, he will have read/write access only to the `default` project, and read only access to the `iris` project.

5.1.13.2 Confirm policies are working

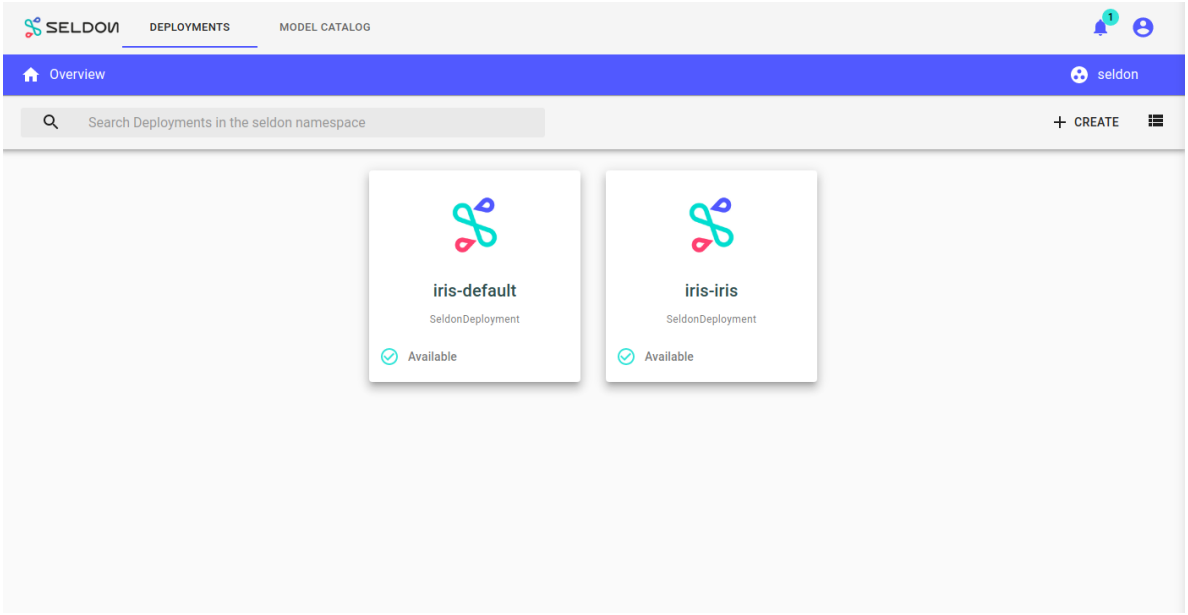
1. Login as `alice`.

2. Go to the `Model Catalog` page and create the following models:
    1. URI: `gs://seldon-models/sklearn/iris`, Project: `default`, Artifact Type: `SciKit Learn`
    2. URI: `gs://seldon-models/sklearn/iris`, Project: `iris`, Artifact Type: `SciKit Learn`
    3. URI: `gs://seldon-models/sklearn/income/model-0.23.2`, Project: `income`, Artifact Type: `SciKit Learn`

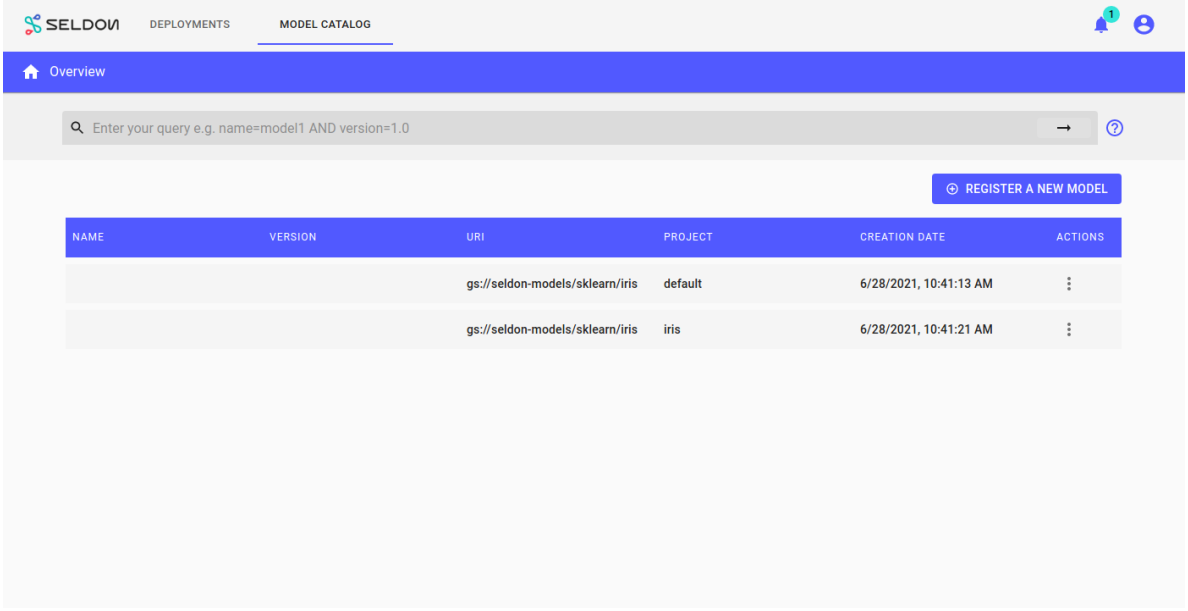
3. Create a deployment from each of these models using the `Deploy` functionality from the Model Catalog. Confirm all three deployments are shown in the `Deployments` tab.

4. Logout from the `alice` profile and login as `bob`.

5. Confirm you only see the `default` and `iris` deployments, but not the `income` deployment.



6. Confirm you only see the `default` and `iris` models, but not the `income` one.



7. Confirm that you cannot delete or modify the `iris` model since `bob` has only `read` permissions on the `iris` project.

5.2 KFServing Demos

These demos assume you have a running Seldon Deploy installed with relevant permissions to create/update deployments.

They mostly use pre-built models, except the kubeflow demo which has steps to build a model and push it to minio.

Models can be pushed to minio or other objects stores for pre-packaged model servers or a custom docker image. See the KFServing docs for the building and hosting stages.

However you install your InferenceServices, Seldon Deploy should see them in visible namespaces. These demos use the Deploy UI for ease.

5.2.1 Canary Promotion

In this demo you will:

- Deploy a SKLearn KFServing Iris Classifer
- Run a load test to show Knative autoscaling
- Create a canary XGBoost model
- Promote the canary
- Delete the model

From the Deploy UI create a server:

Use the model uri:

```
gs://kfserving-samples/models/sklearn/iris
```

5.2.1.1 Run a Load Test

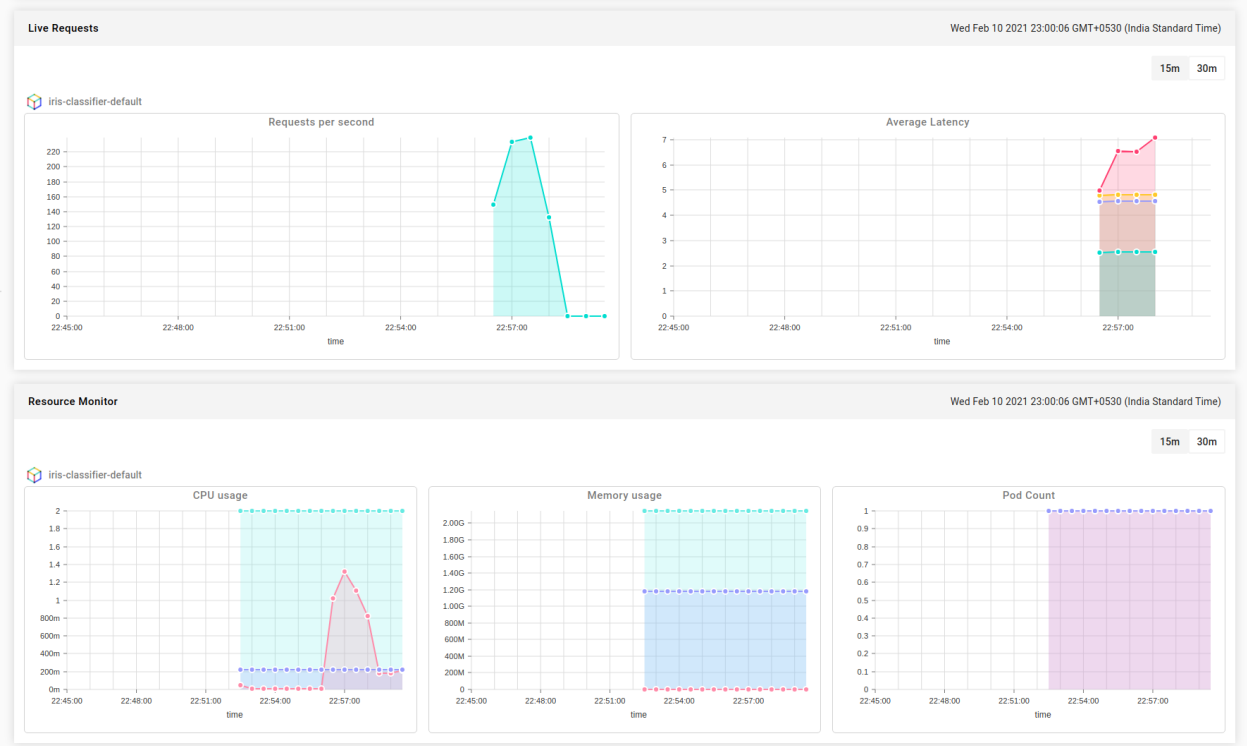
When the deployment is ready click on it and "Start a Load Test".

Set the duration to 60 secs and the number of connections to 10.

Use the `request.json` file in this folder:

```
{
  "instances": [
    [6.8,  2.8,  4.8,  1.4],
    [6.1,  3.4,  4.5,  1.6]
  ]
}
```

You should see pod counts scale up and then down, something like:



5.2.1.2 Adding a Canary

Now follow the "Add Canary" wizard and add an XGBoost canary:

Use the XGBoost Iris model whose saved Booster is stored at:

```
gs://kfserving-samples/models/xgboost/iris
```

Once the canary is running you can rerun the load test and see traffic split between both.

To promote canary press the "Promote Canary" button

5.2.1.3 Delete the Deployment

Finally, you can delete the model.

5.2.2 Model Explanations with Anchor Images

In this demo we will:

- Launch an income classification model which has image training features
- Send a request to get a prediction
- Create an explainer for the model
- Send the same request and then get an explanation for it

This model provides a model trained to classify images based on CIFAR10 dataset.

The explainer uses the *anchors technique* to provide insight into why a particular classification was made by the model. We'll see patterns in an input image that are most relevant to the prediction outcome.

5.2.2.1 Create Model

Use the following model URI with tensorflow runtime.

```
gs://seldon-models/tfserving/cifar10/resnet32
```

5.2.2.2 Get Predictions

Run a single prediction using the tensorflow payload format of an image `truck`.

5.2.2.3 Add an Anchor Images Explainer

Create a model explainer using the URI below for the saved explainer.

```
gs://seldon-models/tfserving/cifar10/explainer-py36-0.5.2
```

5.2.2.4 Get Explanation for one Request

View all requests and then explain it using the JSON below:

5.2.3 Model Explanations with Anchor Tabular

In this demo we will:

- Launch an income classification model which has tabular training features
- Send a request to get a prediction
- Create an explainer for the model
- Send the same request and then get an explanation for it

This model provides a model trained to predict high or low income based on demographic features from a 1996 US census.

The explanation will offer insight into why an input was classified as high or low. It uses the *anchors technique* to track features from training data that correlate to category outcomes.

5.2.3.1 Create Model

Use the model uri:

```
gs://seldon-models/sklearn/income/model
```

5.2.3.2 Get Predictions

Run a single prediction using the JSON below.

```
{"instances":[[39, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]]}
```

5.2.3.3 Add an Anchor Tabular Explainer

Create a model explainer using the URI below for the saved explainer.

```
gs://seldon-models/sklearn/income/explainer-py36-0.5.2
```

5.2.3.4 Get Explanation for one Request

Resend a single request and then explain it using the JSON below:

```
{"instances":[[39, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]]}
```

Currently the display does not show feature names as they've not been supplied for the KFServing request. That is a planned future feature for KFServing. See the equivalent *Seldon demo* for explanations with feature names and for more detail.

5.2.4 Model Explanations with Anchor Text

In this demo we will:

- Launch a movie sentiment model which takes text input
- Send a request to get a sentiment prediction
- Create an explainer for the model
- Send the same request and then get an explanation for it

The explainer uses the *anchors technique* to provide insight into why a particular classification was made by the model. We'll see patterns in input text that are most relevant to the prediction outcome.

5.2.4.1 Create Model

Use the model uri:

```
gs://seldon-models/sklearn/moviesentiment
```

5.2.4.2 Get Predictions

Run a single prediction using the JSON below.

```
{
  "instances": [
   "a visually exquisite but narratively opaque and emotionally vapid experience of␣
↪style and mystification"
   ]
}
```

5.2.4.3 Add an Anchor Text Explainer

Create an Anchor Text explainer using the default settings.

5.2.4.4 Get Explanation for one Request

Resend a single request using the JSON below and then explain it:

```
{
  "instances": [
   "a visually exquisite but narratively opaque and emotionally vapid experience of␣
↪style and mystification"
   ]
}
```

5.2.5 Model Catalogue

5.2.5.1 Registering Models and Editing Metadata

Notes:

1. Model Catalog is currently experimental.
2. This feature requires *PostgreSQL to be installed*.
3. This demo expects that there is no existing model with the same `URI` or the same combination of `Model Name` and `Version`. The model catalog enforces uniqueness on these fields. Requests that result in a conflict will be rejected.

5.2.5.2 Register New Model

1. From the Deployments Overview page, select `Model Catalog` at the top of the page.
2. Click `Register A New Model` to open the registration modal.
3. Enter the following parameters:

- *Model Name*: `cifar10`
- *URI*: `gs://seldon-models/tfserving/cifar10/resnet32`
- *Artifact Type*: `Tensorflow`
- *Version*: `v1.0`
- *Task Type*: `classification`

1. Append tags and metric parameters with the following values. To add more parameters, use the + button. To delete a parameter, use the `x` button:

- Tags:
  - *key*: `author`, *value*: `Seldon`,
  - *key*: `training_set`, *value*: `Resnet32`,
- Metrics:
  - *key*: `p1`, *value*: `0.8`,
  - *key*: `p2`: *value*: `0.6`

1. Click `Register Model`

5.2.5.3 Edit Model Metadata

From The Model Catalog

1. From the Deployments Overview page, select `Model Catalog` at the top of the page.
2. Select a model. For the purposes of this demonstration, the model registered in the Register New Model step will be used.
3. On the side drawer that opens, click `EDIT METADATA`.
4. Add a new tag with the following values:

- *key*: `stage`, *value*: `production`

1. Click `SAVE METADATA` at the top right hand side of the side drawer to save your edit.

From A Deployment's Dashboard

Note: Please follow the instructions on creating an *Outlier Detector*.

1. From the Deployments Overview page, select the deploment to append metadata on the running model. For the purposes of this demonstration, the deployment created from the *Outlier Detector* example will be used.
2. From your selected deployment's dashboard, click the model being used to open a drawer on the right.
3. Click `EDIT METADATA`.
4. Under the *Tags* section, press the + button to add a new tag.
5. For the new tag, enter the following parameters:

- *key*: `stage`, *value*: `production`

1. Click `SAVE METADATA` on the top right of the side drawer.

Deploying Models From The Model Catalog

Note: This demonstration requires the cifar10 model to have been registered already.

1. From the overview page, navigate to the model catalog using thr header tabs.
2. Select the *more* icon for the model named `cifar10`.
3. In the dropdown menu that appears, select *Deploy*.
4. In the wizard which appears, name the *deployment* `cifar10`, and set the protocol to `KFServing`. Click `next` when done.
5. On the *Default Predictor Parameters* screen, the model URI should automatically be appended. For Triton models, the *model name* will also be appended automatically. Click *next* to continue.
6. Skip the remaining steps and click `Launch`.

Select `Deployments` header tab to verify that your model has been deployed.

5.2.6 Outlier Detection with CIFAR10 Image Classifier

This demo is based on VAE outlier detection in the alibi detect project. Here we will :

- Launch an image classifier model trianed on the CIFAR10 dataset
- Setup an outlier detector for this particular model
- Send a request to get a image classification
- Send a perturbed request to get a outlier detection

Important: This demo requires Knative installation on the cluster as the outlier detector will be installed as a kservice.

5.2.6.1 Create Model

Use the following model uri

```
gs://seldon-models/tfserving/cifar10/resnet32
```

5.2.6.2 Setup Outlier detector

Setup an outlier detector with model name `cifar10` using the default settings (which sets Reply URL as seldon-request-logger in the logger's default namespace - change if you modified this at install time) and storage URI as follows:

```
gs://seldon-models/alibi-detect/od/OutlierVAE/cifar10
```

5.2.6.3 Make Predictions

Run a single prediction using the tensorflow payload format of an image `truck`. Also a perturbed image of the truck in the same format at `outlier truck image`. Make a couple of these requests at random using the predict tool in the UI.

5.2.6.4 View outliers on the Requests Screen

Go to the requests screen to view all the historical requests. You can see the outlier value on each instance. Also you can highlight outliers based on this score and also use the filter to see only the outliers as needed.

5.2.6.5 Monitor outliers on the Monitor Screen

Under the 'Monitor' section you can see a timeline of outlier requests.

5.2.7 Drift Detection with CIFAR10 Image Classifier

This demo is based on model distiliation drift detector in the alibi detect project.

The idea is that input data can change over time and become significantly different from the model training data. When the input data distribution shifts then prediction quality can drop. We want to know when that happens.

Here we will :

- Launch an image classifier model trianed on the CIFAR10 dataset.
- Setup a drift detector for this particular model.
- Send a request to get a image classification.

- Send a request to trigger a drift detection.
- View the drift in the dashboard.

Important: This demo requires Knative installation on the cluster as the outlier detector will be installed as a kservice.

5.2.7.1 Create A Model

Note

1. Verify the namespace you will be deploying to.
2. Deployment names must be unique and only contain alphanumeric characters.

From the *deployment overview* screen, click on the Create button to create a new deployment.

1. In the deployment creation wizard, enter a name for your new deployment.
2. Select the namespace you would like the deployment to reside in *(e.g. `seldon`)*.
3. From the *protocol* dropdown menu, select `Tensorflow` and click next.
4. For the deployment details, enter the following values, then click next:
   - Runtime: `Tensorflow`
   - Model URI:

   ```
   gs://seldon-models/tfserving/cifar10/resnet32
   ```
5. Skip the remaining steps, then click `Launch`.

5.2.7.2 Add A Drift Detector

Note

1. Make sure you are in the correct namespace.
2. By default, the Reply URL is set as `seldon-request-logger` in the logger's *default namespace*. **If you are using a custom installation, please change this parameter according to your installation**.
3. Increase this value to make the drift detector less sensitive.

From the *deployment overview page*, select your `cifar10` deployment to enter the deployment dashboard.

Inside the *deployment dashboard*, add a drift detector with by clicking the `Create` button within the `Drift Detection` widget.

Enter the following parameters in the modal popup which appears, using the default settings:

- *Model Name*: `cifar10`.
- *Model URI*:

  ```
  gs://seldon-models/alibi-detect/cd/ks/cifar10-0_4_4
  ```
- *Reply URL*:

```
http://seldon-request-logger.seldon-logs
```

- *Batch Size*: `2`.
- *Protocol*: `Tensorflow`.
- *HTTP Port*: `8080`.

Then, click `Create Drift-Detector` to complete the setup.

5.2.7.3 Make Predictions

1. From the deployment dashboard, click on `make new prediction`.
2. Run a single prediction using the tensorflow payload format of an image `truck`.

Next, run a single prediction using the tensorflow payload format of an image of an `outlier truck image`.

5.2.7.4 Monitor Drift On The Monitor Screen

Under the 'Monitor' section of your deployment, you can see a timeline of drift requests.

This becomes clearer if you make many requests over a period of time.

5.2.8 Distributions Monitoring

This demo uses a model trained to predict high or low income based on demographic features from a 1996 US census. In this demo we will observe the predictions and feature distributions of live predictions made using this model with the following steps:

- Register an income classifier model with the relevant predictions schema
- Launch a KFServing deployment with the income classifier model
- Make predictions using a REST requests to the model deployment
- Observe the feature distributions of the live predictions
- Filter distributions by time or predictions and feature level filters

---

Note: This demo needs the request logger to connect to Seldon Deploy in order to fetch model level predictions schema. And this requires specific request logger configuration. Also this feature is supported with many protocols available with deployments like seldon, tensorflow and the kfserving v2 protocol. But not supported for json data, string data, bytes payload or multi-node graph use cases yet.

5.2.8.1 Register a income classifer model

Register the income classifier SKLearn model with the below URI.

```
gs://seldon-models/sklearn/income/model
```

5.2.8.2 Configure predictions schema

Edit the model metadata to update the predictions schema for the model. The predictions schema is a generic schema structure for machine learning model predictions. It is a definition of feature inputs and output targets from the model prediction. Use the `income classifier model predictions schema` to edit and save the model level metadata. Learn more about the predictions schema at the ML Predictions Schema open source repository.

5.2.8.3 Launch a KFServing deployment

Deploy the income classifier model from the catalogue into an appropriate namespace.

5.2.8.4 Make predictions using the model deployment

Model predictions can be made in the appropriate protocol. In this demo, we will run several single predictions payload examples below:

```
{ "instances": [[39, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]] }
{ "instances": [[50, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]] }
{ "instances": [[23, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]] }
{ "instances": [[21, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]] }
{ "instances": [[25, 7, 1, 1, 1, 1, 4, 1, 2174, 0, 40, 9]] }
```

Distributions monitoring is especially useful to keep track of predictions when a model makes thousands of predictions in real world scenario. To simulate such a use case, make multiple predictions over time in the Seldon protocol request format using the `predictions data csv file` and the following shell script which makes around `32560 predictions with an interval of 5 seconds between each request`. Note that you need sufficient payload logging infra needed for this.

```
CLUSTER_IP=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}')
SERVICE_HOSTNAME=$(kubectl -n seldon get inferenceservice kf-income-classifier -o␣
↪jsonpath='{.status.url}' | cut -d "/" -f 3)
DEPLOYMENT_NAME=kf-income-classifier
INTERVAL_SECONDS=5

while IFS= read -r line; do
    curl -k -H "Content-Type: application/json" -H "Host: ${SERVICE_HOSTNAME}" \
        http://$CLUSTER_IP/v1/models/$DEPLOYMENT_NAME:predict \
        -d '{ "instances": [['${line// /,}']] }'
    sleep "${INTERVAL_SECONDS}s"
done <prediction-data.csv
```

5.2.8.5 Observe predictions and feature distributions

Select the income classifier deployment and go to the monitor section to view the predictions and feature distributions.

5.2.8.6 Filter distributions by time or feature level filters

Filter distributions by time or predictions and feature level filters to compare different cohorts and further analysis. For example lets look at the predictions for all individuals in the `Age` group 25-45 and see how the average prediction frequency changes for this cohort.

5.2.8.7 Configuring parameters

Distributions parameters configuration allows you to configure your charts for further analysis. For example let's look at at the charts in the `Age` group and change the `Histogram interval` to `11` and `Number of time buckets` to `30` to see.

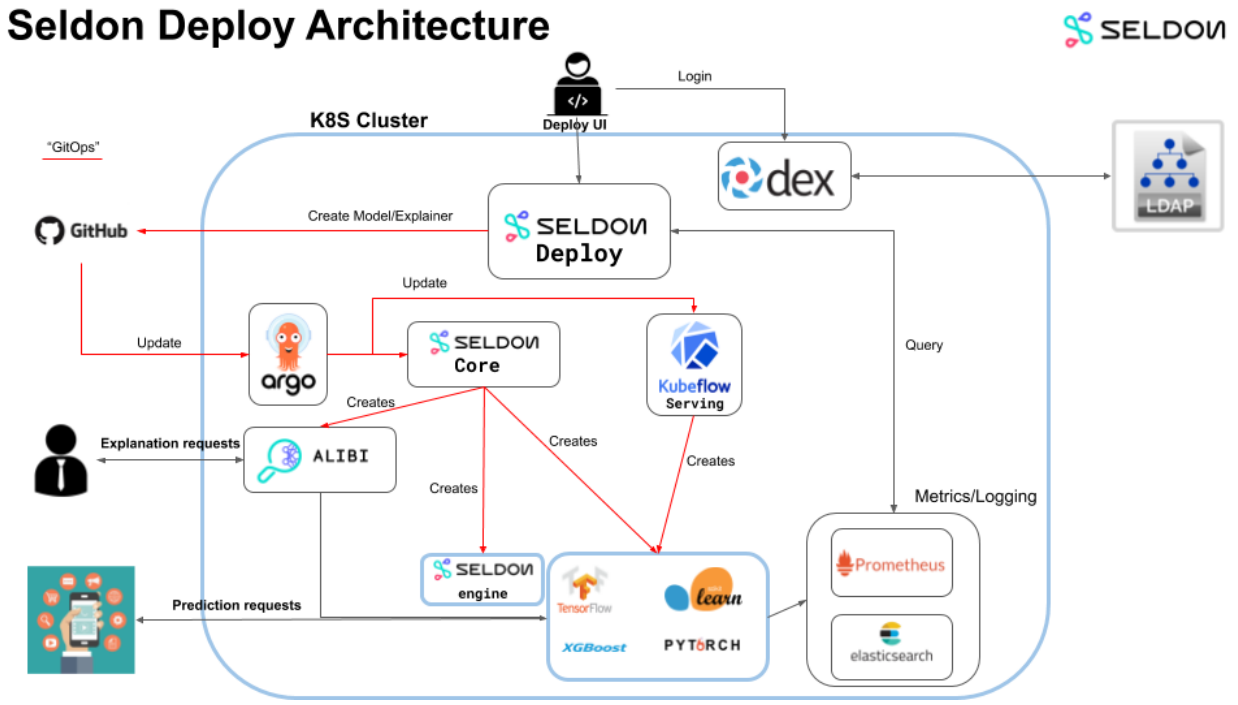
CHAPTER

SIX

ARCHITECTURE

Overview of the technical architecture

Seldon Deploy is made up of several interacting components that the application uses. The core design is illustrated in the diagram below.



Seldon Deploy manages the running of our open source core components Seldon Core, KFServing, and Seldon Alibi. It also manages a range of over components to provide a full machine learning deployment platform.

The following sections will discuss the core parts of the architecture.

6.1 Authentication

User Authentication

Currently, Seldon Deploy SSO can be configured in two ways.

- kubeflow cluster
- *app-level authentication*

6.1.1 App-level Auth

App-level auth is where a request to Seldon Deploy is checked for an auth token and if that is missing then Deploy redirects to the auth system.

Deploy uses OIDC and can be integrated to OIDC auth systems.

6.1.2 Kubeflow Gateway Auth

In the kubeflow architecture the auth is done at gateway level, before reaching Deploy or any other apps specifically:

This 'existing_arrikto' architecture uses an istio envoy filter to run all requests through a check for an auth token, performed by the OIDC AuthService. If no token is available then the user is sent to dex to login and obtain a token.

LDAP, or another external identity provider, can be used as per the existing_arrikto example and in line with the dex documentation.

In order to enforce restrictions, namespaces for Seldon Deploy are labelled as restricted or unrestricted and with the operations that groups can perform on them. See `Deployments by Namespace` under the `Product Tour` heading.

6.2 Authorization

User Authorization

Once a user is *authenticated* their requests can be authorized based on their permissions. The user information from the OIDC token is used to identify the user by their ID and the groups they belong to. Permissions can be defined both on user and on group level.

Resources in Deploy are grouped into two categories:

- Namespace based resources - these are resources that belong to a specific Kubernetes namespace. This includes all Seldon Deployments and Inference Services running in Deploy. All endpoints and UI actions that read or modify a namespaced resource can be authorized based on the permission of the user to that namespace.
- Project based resources - these are the models stored in the *Model Catalog* and deployments using these models.

Currently, there are two ways of defining permissions on the various resources served by Seldon Deploy.

6.2.1 Kubernetes Namespace Labels

As described on the namespace visibility page, a Kubernetes namespace can be labelled to denote who has access to it. Seldon Deploy will check if the user requesting a namespaced resource is given the required permissions by the namespace labels before serving the request.

Namespace labels are used only for authorizing requests to namespace based resources. For a more complete authorization solution, check the *Open Policy Agent policies*.

6.2.2 Open Policy Agent policies

Seldon Deploy can use Open Policy Agent (OPA) policies to determine if a user has access to a resource. OPA is popular open-source technology for defining flexible cloud-native policies. To enable it follow the *installation guide*. This is an experimental feature.

OPA policies can be used to authorize both namespace based resources and *project based resources*.

OPA policies will become the primary way of authorization in future releases. We are working on improving the support and functionality around IAM. If you have any feedback please reach out to your Seldon representative.

6.3 Explainers

Machine Learning Model Explanation

Seldon uses its Open Source Alibi Model Explanation library to provide model explanation. The library provides a wide range of techniques which will be integrated and optimized for usage in Seldon Deploy.

6.4 Gateways

Seldon Core and KFServing both have features for traffic-splitting. For this they rely on a Gateway.

Currently Seldon Core supports Istio or Ambassador. KFServing currently only supports Istio.

Detector components and KFServing both also use Knative. That requires one of requires one of Ambassador, Contour, Gloo, Istio, Kong or Kourier

Seldon Deploy itself does not have a hard dependency on Istio. It has dependencies on components that use Istio.

Seldon Deploy can be used to make predict calls to models. The gateway affects the form of the urls to the models. However, this is configurable in the Seldon Deploy values file using templating to set the form of the url from relevant parameters. It defaults to Istio.

The key issues to consult on the upstream projects are the below - the Knative one at the top explains the situation well:

https://github.com/knative/serving/issues/10417 https://github.com/SeldonIO/seldon-core/issues/2988 https://github.com/SeldonIO/seldon-core/issues/416 https://github.com/SeldonIO/seldon-core/issues/1129

Please comment on the above or otherwise contact Seldon if this is of interest.

6.5 GitOps

Git for Source Control

Seldon Deploy will scan namespaces and recognise some namespaces, those with a `seldon.gitops: enabled` label, as GitOps namespaces. If the GitOps label is not present or disabled then new deployments and modifications in the namespace will be pushed directly to the kubernetes cluster.

When GitOps is in place it is recommended to have a GitOps tool syncing a namespace to a directory in a git repo. The install has an example setup for argocd.

GitOps responsiveness improves if webhooks are configured. One webhook should be configured to notify the GitOps tool (e.g. argocd) so that it can update the cluster on changes as quickly as possible. Another should be configured to inform Seldon Deploy of any changes to the repository.

6.6 Kubeflow

End to End Machine Learning on Kubernetes

We integrate with the Kubeflow project to allow end to end machine learning.

- Utilise Seldon and KFServing products from kubeflow.
- Integrate deployment into Kubeflow Pipelines.

6.7 Model Metadata Store

Storage, APIs, and UI for managing model metadata

Seldon Deploy provides a storage solution for model's metadata since v1.2. It allows model information to be stored and referenced in Seldon Deploy.

The model metadata is stored in Postgres, which can be part of the Kubernetes cluster, or a managed solution. For more information on how to setup the Postgres configuration in Seldon Deploy see *the instructions*.

Seldon Deploy watches for changes to the cluster and keeps track of what models are used in what deployments. You can access this data from the API using the `/api/v1alpha1/model/metadata/runtime`. In future releases we will make this information available through the UI as well allowing filtering deployments by model metadata.

Seldon Deploy also prepopulates basic model information for all models already running in the cluster that are not yet registered in the Model Store.

6.7.1 Querying model metadata

6.7.1.1 Making queries

Seldon Deploy provides an advanced query language that allows you to make sophisticated search queries over your models.

Fields can be compared using the following operators: `=, !=, >=, <=, <, >`, and multiple queries can be combined using `AND, OR`.

For example you can search for all models with the name `iris` and version `2.0` with a query like this:

```
name=iris AND version=2.0
```

More advanced queries can be built up using bracketed expressions:

```
(name=iris AND version=2.0) OR version=3.0
```

Note that since the version field is a string, the `>=, <=, <, >` operators are alphabetical comparisons, not numeric ones.

6.7.1.2 Querying for tags and metrics

You can query for models that have specific metrics or tags by using the following syntax:

```
metrics[metricName]>1.0 AND tags[tagKey]!=someValue
```

6.7.1.3 Queries using the API

Using the model metadata API you can specify a string query (the same way you would in the UI) which takes precedence over other fields set in the API.

```
<yourSeldonDomain>/seldon-deploy/api/v1alpha1/model/metadata?query=(version=1.
0)
```

6.7.1.4 Valid query field names and values

Field Names	Valid Values
URI	string
Name	string
Version	string
Project	string
Arti-fact_type	one of `CUSTOM, TENSORFLOW, SKLEARN, XGBOOST, MLFLOW, PYTORCH, ONNX, TENSORRT`
Task_type	string
Tags	`tags[tag_name]=tag_value` (string data)
Metrics	`metrics[metric_name]=metric_value` (numeric data)

Note that the field names are case insensitive.

6.8 Monitoring

Real time observability for ML models

Seldon Core and KFServing both provide out of the box grafana dashboards for monitoring. These include metrics on requests, feedback on predictions and request latency. The dashboards can be used to compare different versions of a model between which traffic is being split, such as a main model running alongside a canary version. These dashboards are integrated into the Seldon Deploy user interface and are accessed behind the same authentication as the rest of Seldon Deploy. By default this is achieved by exposing a single, secured istio ingress gateway and creating VirtualServices for each component (other configurations would also be possible).

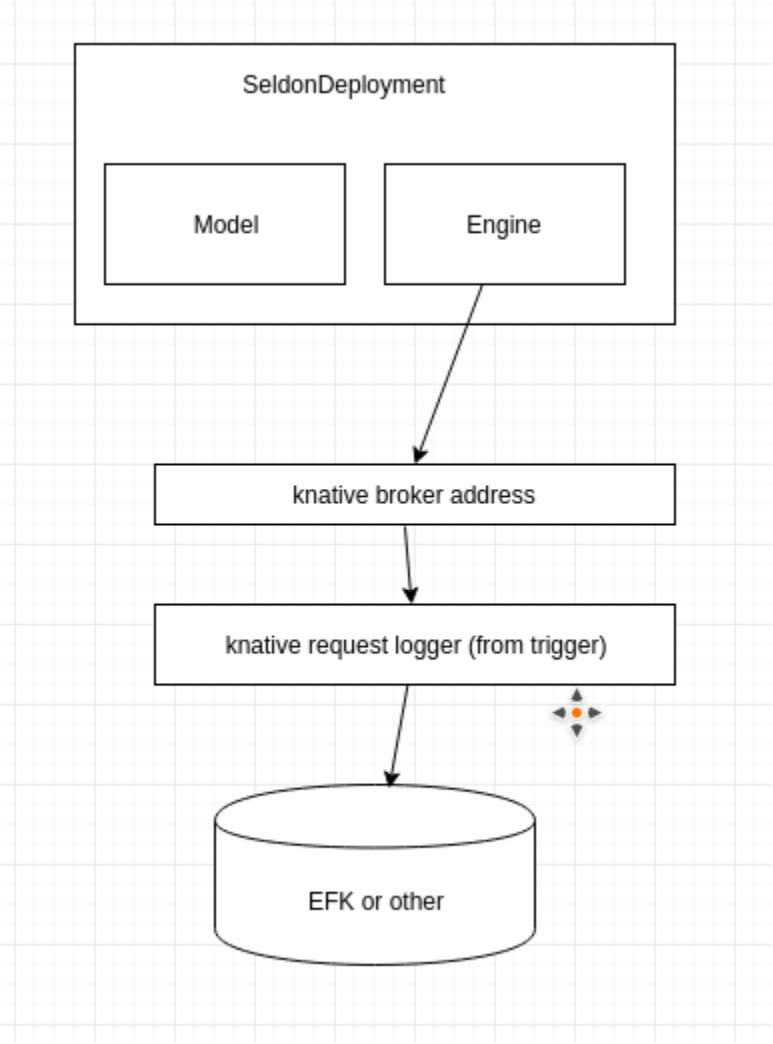
Detailed demos showing monitoring are available in the Demos section of this documentation.

6.9 Request Logging

Log prediction requests and responses for analysis

In Seldon Core request logging is initiated by the seldon container engine. It sends a request-response pair to a knative broker (part of knative eventing) so that it can be processed asynchronously.

The knative broker has a listening trigger which can bring up a request logger. By default the request logger logs to stdout for fluentd to collect and send to elasticsearch, which is the primary backend integration:



The request logger component labels requests to distinguish text, tabular and image data. It also splits out batch requests so that they are individually logged. This allows requests to be displayed in the UI and selected for running explanations.

Request logging for KFServices is being tracked in github

6.10 Serving Solutions

We support the following serving technologies

6.10.1 Seldon Core

Seldon Core can be installed via a helm chart, kustomize or as part of kubeflow. For full use of Seldon Deploy features it should be installed with istio routing and request logging enabled.

SeldonDeployment custom resources are managed by the seldon core operator, typicall installed in the seldon-system namespace.

When a SeldonDeployment is deployed the operator creates the necessary Kubernetes pods. One of these pods will contain the seldon-container-engine, which reads the inference graph and orchestrates the HTTP calls to follow the sequence of the graph components.

Monitoring is via seldon-core-analytics, which is a grafana install that comes with OOTB dashboards.

Details on the request logging stack can be found in the Request Logging section.

6.10.2 KFServing

KFServices are handled by the KFServing operator. This controller-manager is installed by default in the kubeflow namespace as part of a kubeflow install.

Services are implemeted as knative services using knative serving. Monitoring is via the knative-monitoring stack.

Inference graphs are not currently part of KFServing but see the project roadmap

CHAPTER

SEVEN

OPERATIONS

Operating Seldon Deploy in your cluster

These pages go into detail on best practices for operators or admins managing a Seldon Deploy cluster.

7.1 App Analytics

Seldon Deploy has built-in app analytics, which help to improve the product. These analytics focus on tracking certain events that get triggered from the UI (e.g. user login).

Note: All events share a `distinct_id` attribute. This attribute is **linked to the IP address**, so that each IP address is identified as a unique user.

7.1.1 Disable app analytics

You can disable the app analytics through the `enableAppAnalytics` variable of the Seldon Deploy Helm chart. To disable it just add the following to your Helm variables:

```
# boolean to enable app-analytics (defaults to "true")
enableAppAnalytics: false
```

7.1.2 Events tracked

Name	Trigger	Attributes
`user_login`	Show homepage after logging in	User login attributes

7.1.2.1 User login attributes

Property	Example Value	Notes
`BuildTime`	Wed Apr 22 15:59:09 UTC 2020	
`BuildVersion`	0.6.0	
`GitBranch`	master	
`GitCommit`	5ad34e2b556732213873ee6875f2f99d20a471ee	
`GitStatus`	dirty	
`ReleaseType`	Early-Access	
`distinct_id`	ABABABABABABBABAB	Linked to IP address

7.2 Deployment Models

Deployment models for Seldon Deploy

When it comes to deploying Seldon Deploy for a production workload, there are different deployment models that can apply depending on your use case. For example, you can consider the following questions:

- Do you want to support multiple environments (e.g. `production`, `staging`, etc.)? Will these be split across multiple clusters? Or across different namespaces?
- Are you planning on supporting multiple tenants in your infrastructure? How will you manage permissions for each one of them?

This section will showcase some of the most common deployment models to set up Seldon Deploy in your cluster.

7.2.1 Multiple environments

When you are managing machine learning models, a common use case is to test them first on some kind of test environment, isolated from production workloads. These test environments are usually known as `staging`, `qa`, `pre-prod`, etc.

In some cases, these environments will form a deployment (or continuous delivery) pipeline, where each environment is considered more stable than the previous one:

```
Dev > ... > Staging > Production
```

Seldon Deploy allows you to have multiple environments by separating them as namespaces or clusters.

Note: Regardless of the approach you choose, this doesn't restrict your options when it comes to syncing your models with a GitOps repository. In fact, you can still have:

- Single GitOps repository, with one folder per environment.
- Multiple GitOps repositories, with one repository per environment.
- ...

7.2.1.1 Namespace environments

The official Kubernetes docs define namespaces as *virtual clusters*. Therefore, it makes sense to treat each separate environment as a separate namespace. Following the example above, that would mean creating a `dev`, `staging` and `production` namespaces.

By default, Seldon Deploy will treat each namespace as a separate environment. Therefore, this is the **suggested approach**.

Out of the box, this gives you easy control over:

- Where is each model deployed (on which environments).
- Who is *authorised to view or create models*.
- Visibility over all your models across every environment from a single instance of Seldon Deploy.

7.2.1.2 Cluster environments

When you are dealing with multiple environments, it makes sense to keep them completely isolated of each other. That way, any cluster-wide effects will only affect one of them.

Using separate Kubernetes clusters for each of your environments allows you to achieve a higher-level of separation than namespaces. However, this separation also comes with a few caveats:

- All your infrastructure resources will need to be duplicated and managed across every environment (e.g. auth, ingress, logging, metrics, . . .).
- Operations spanning multiple environments become more complicated (e.g. promotion of models between different environments).

This approach is usually recommended to test new versions of cluster-wide components at the infrastructure level. For example, a new version of Seldon Deploy or a new ingress system. However, due to the caveats listed above, we don't suggest this approach to segregate different versions of your models.

Having said so, you can still leverage the GitOps paradigm to communicate between different clusters.

7.3 GitOps Sync

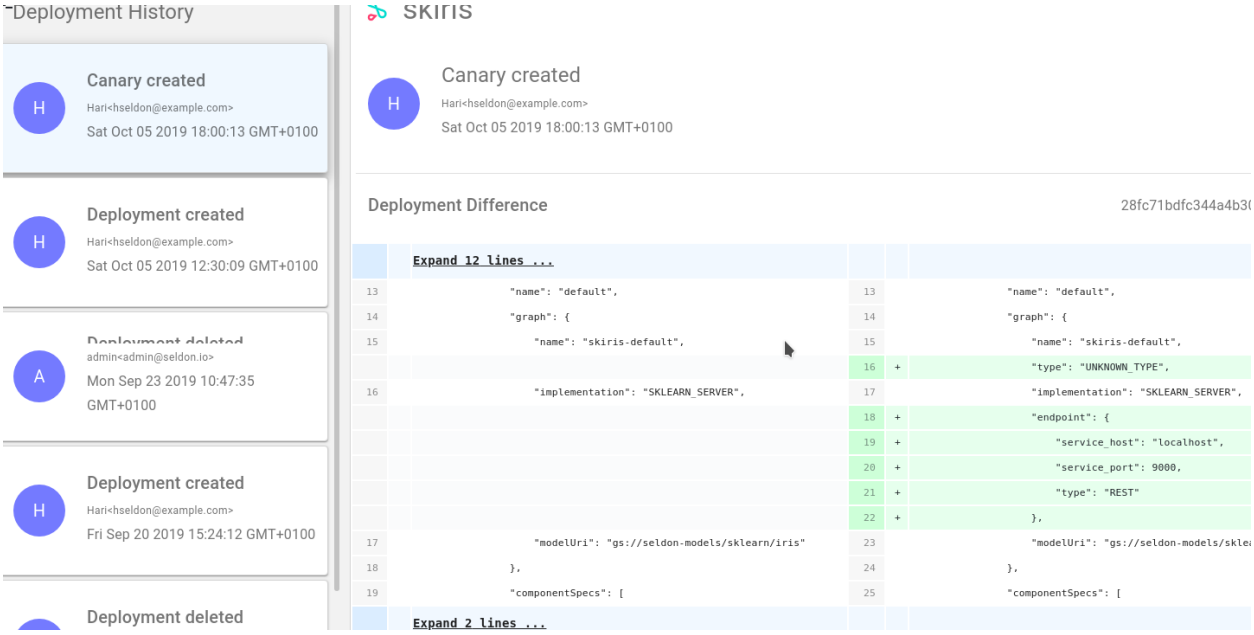
Deployment source control and syncing with Gitops

Seldon Deploy can recognise that certain namespaces are to be maintained using GitOps. It will look for the label `seldon.gitops: enabled`.

If the gitops label is not present or disabled then new deployments and modifications in the namespace will be pushed directly to the kubernetes cluster.

If the gitops label is present then deploy will look for an annotation named `git-repo` and a git URI. It uses a service account token for accessing the repo which is stored in a Secret installed with Seldon Deploy using the helm chart (along with username and email). Deploy will add metadata to any commits it makes, including recording which dex user took the action.

Seldon Deploy can display an `Audit Log` for each deployment where it reads back git commits and makes the changes and metadata visible:



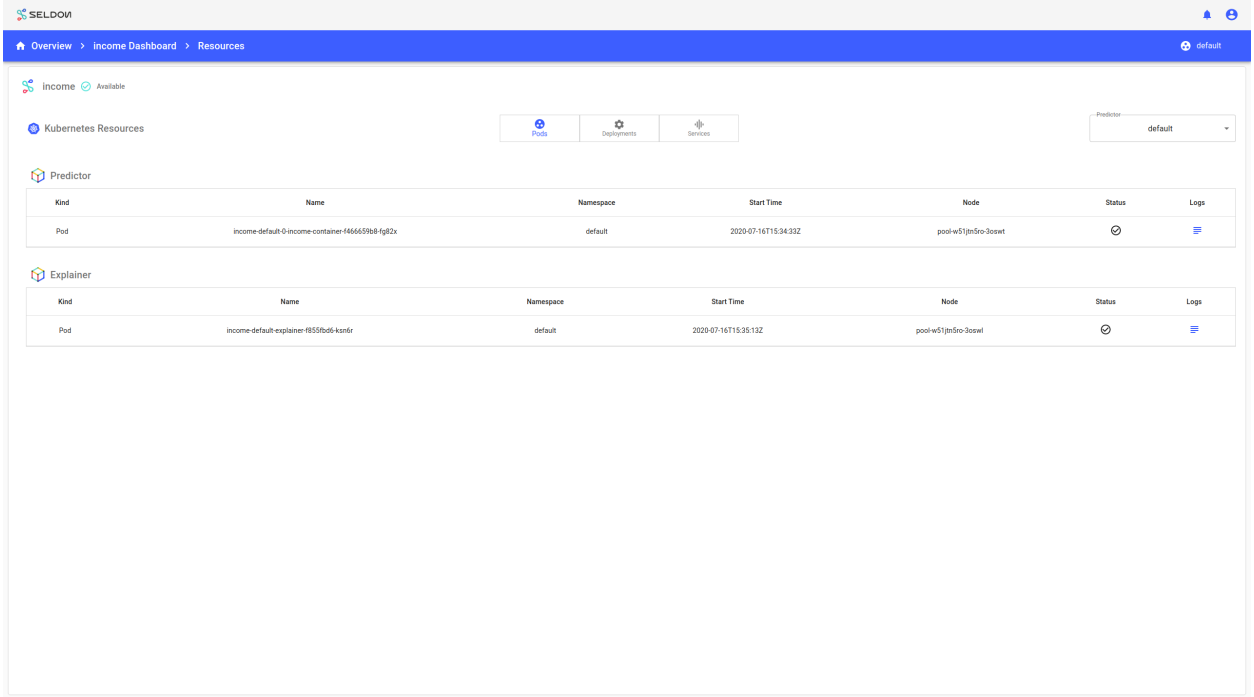
If the user is permitted then the state can also be restored to a previous commit.

See `GitOps` under `Architecture` for more on how GitOps works.

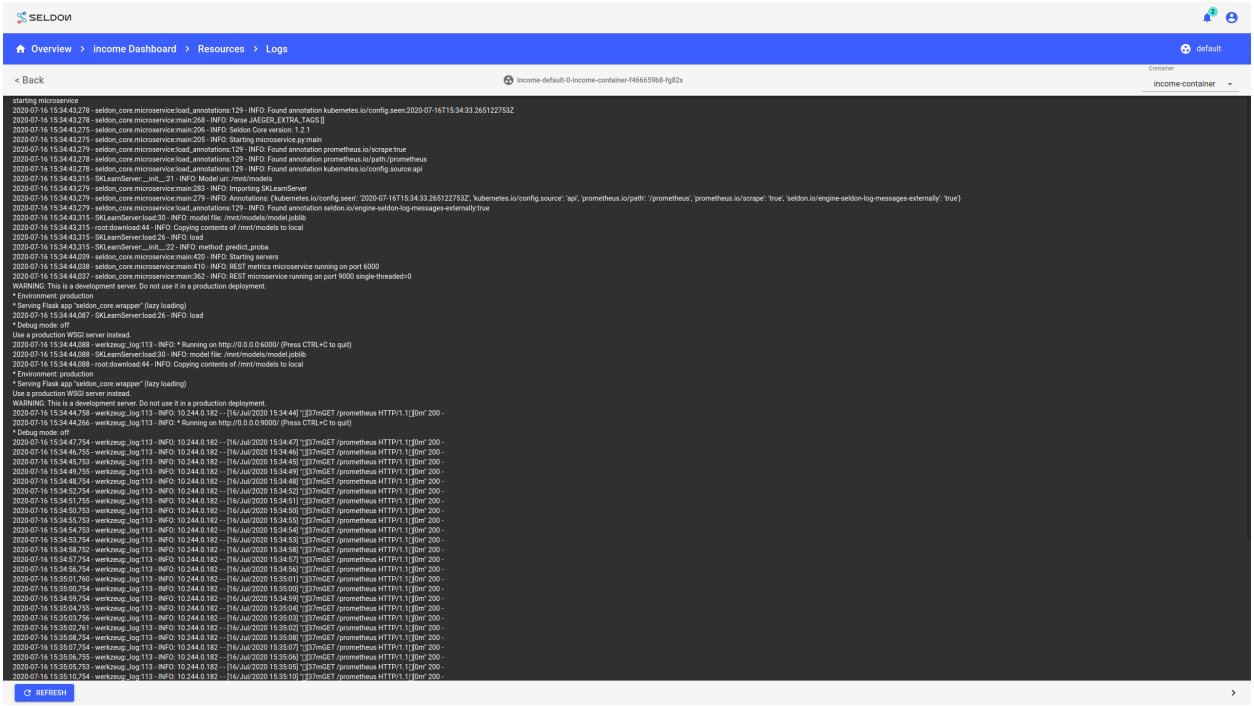
7.4 Kubernetes Resources

Underlying kubernetes resources and debugging

Seldon Deploy allows a view of the underlying kubernetes resources created by the deployed models. This view displays the pods, deployments and services created for each component of the Seldon Deployment/Inference Service custom CRDs.



Also for the pods created for each model component, you can dive into the live container logs and debug relevant issues if any.



7.5 LDAP Integration

7.5.1 Integration Options

Seldon Deploy integrates with Single Sign-On (SSO) providers for authentication.

The SSO system may also host the identities, or it may integrate to an IdP. The IdP is often a source of truth for users across the organisation.

User details can be used for *filtering/restricting visibility of namespaces*. Filtering is either by user or by group.

In order for group-based filtering to be used, groups need to be available. These come from an IdP but not all IdPs support groups or support groups in integration to SSO.

Here is a list from the Dex documentation (see their docs for latest):

Name	supports refresh tokens	supports groups claim
LDAP	yes	yes
GitHub	yes	yes
SAML 2.0	no	yes
GitLab	yes	yes
OpenID Connect	yes	no (#1065)
Google	yes	yes
LinkedIn	yes	no
Microsoft	yes	yes
AuthProxy	no	no
Bitbucket Cloud	yes	yes
OpenShift	no	yes
Atlassian Crowd	yes	yes
Gitea	yes	no

If groups are to be used, we suggest integrating either keycloak or dex to LDAP. See, for example:

- Keycloak LDAP integration example
- Dex LDAP integration example

Or search the official documentation of those products for the latest details.

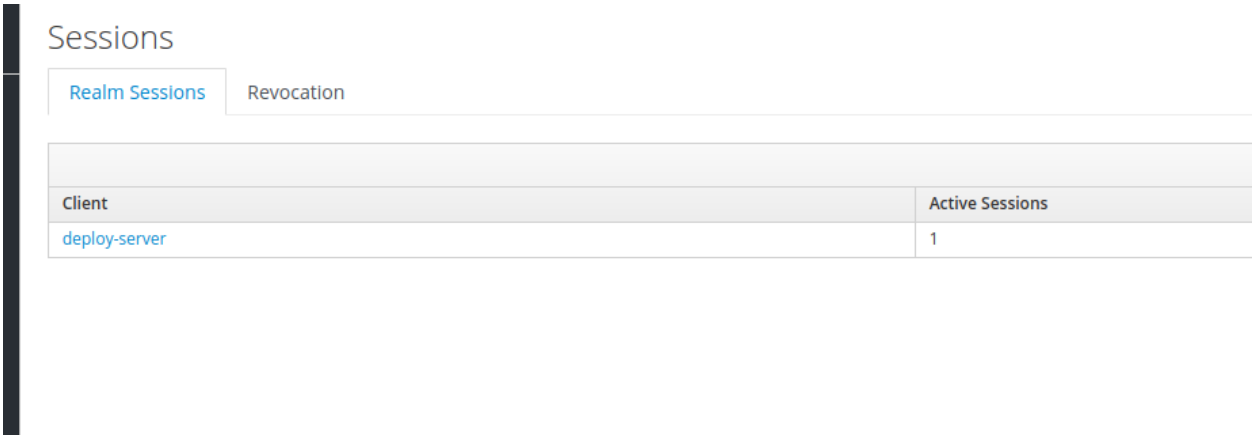
7.5.2 Debugging Keycloak/Groups

Keycloak is the most common choice at present for Identity Provider to integration to LDAP.

If configured correctly you should be able to *filter namespaces by group permissions*)

If you are not sure if groups are coming through, go to the about page in Deploy and see if a group is shown for your user. Here the group is Data Scientist:

Then open another window and login to keycloak as an admin user. Go to 'Sessions':

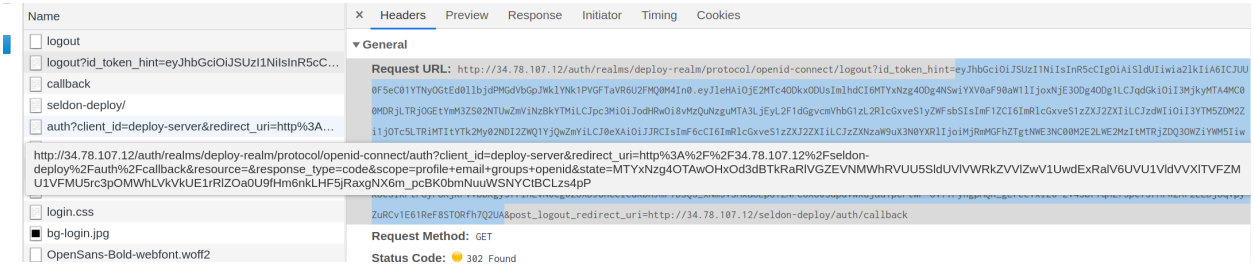


Click into the Session and click 'Show Sessions':

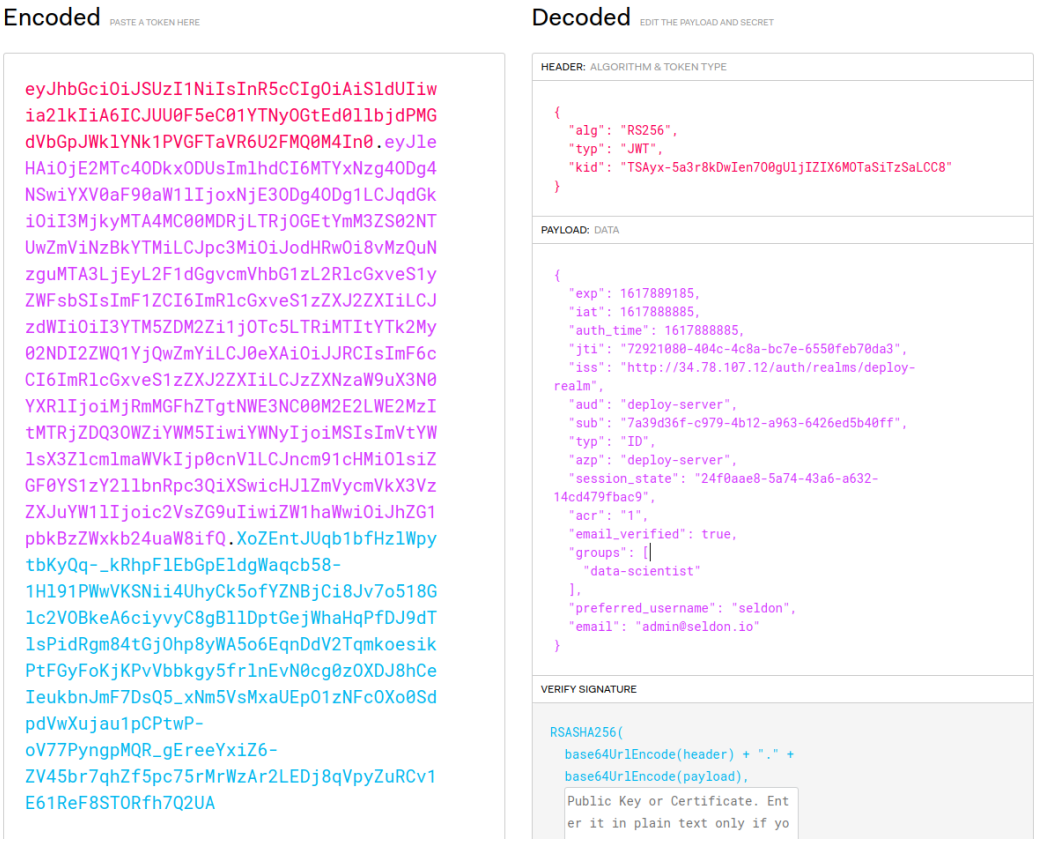
Click into that and then go to Groups:

This will tell you whether the Groups are coming through at a Keycloak level.

You can also get the full token from Seldon Deploy. To do this inspect the browser network tab with preserve logs turned out. Then logout your session. You should see a logout containing the token:



If you put the content of id_token_hint in jwt.io then you can see its contents, including groups.



If keycloak has the groups but the token does not then likely the 'groups' scope is missing in the OIDC_SCOPES env var in the Seldon Deploy helm configuration.

Note that there needs to be a client scope in the keycloak admin console for groups and this needs to be mapped to groups using the group membership mapper type. The scope and token claim name should both be 'groups'. Disable 'full group path'.

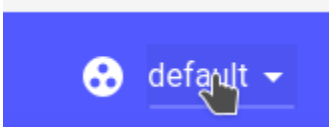
The groups scope has to be added as a client scope under the keycloak client config for the client used by Seldon Deploy.

7.6 Namespace Visibility

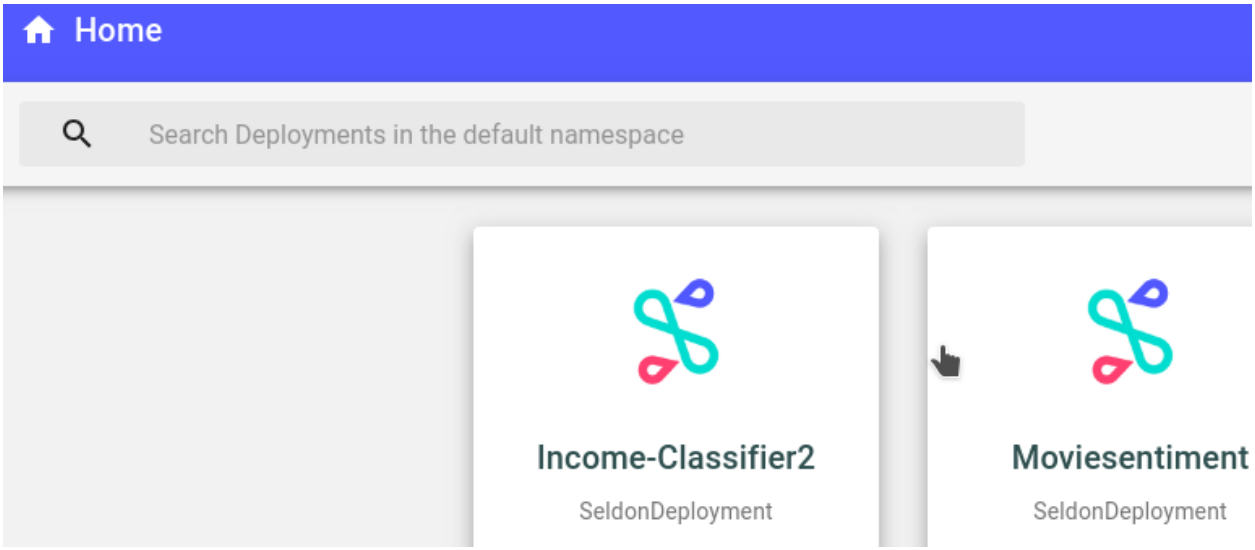
Deployment filtering by namespace

7.6.1 Visibility Restriction per Namespace

Namespaces that the user is eligible to see appear in a drop-down namespace selector in the top-right:



From there the user can search for or choose to drill into deployments:



Visibility and permissions on namespaces are driven by labels or by authorization policies.

7.6.2 Using namespace labels

7.6.2.1 Global Visibility

By default Seldon Deploy does not show any namespaces that are not explicitly labelled.

You can make a namespace globally available to all users by providing the following label to the namespace:

```
seldon.restricted: "true"
```

This indicates that the namespace is restricted. If the label were `seldon.restricted: "false"` then the namespace would be open to everyone. Namespaces without a `seldon.restricted` label would be ignored.

This can be set with `kubectl label namespace $namespace seldon.restricted=false --overwrite=true`, $namespace is the namespace to label.

7.6.2.2 Group Based Visibility Restrictions

For fine-grained restrictions on a group level for namespace access, you can add namespaces with the label in the format:

```
seldon.group.{YOUR_GROUP_NAME}: "write"
```

Some examples of these, which would be linked to your LDAP integration include:

```
seldon.group.admins: "write"
seldon.group.developers: "read"
```

This would indicate that members of the admins group could make changes in the namespace and developers can read. Groups could, for example, come from LDAP. See `Authentication` under `Architecture`.

7.6.2.3 User-Based Visibility Restrictions

For fine-grained restrictions on a user level for namespace access, you can add namespaces with the label:

```
seldon.user.{USERNAME}: "[write/read]"
```

Some examples of these, which would be linked to your LDAP integration include:

```
seldon.user.myusername: "write"
seldon.user.anotherusername: "read"
```

The claim to be used for user-based visibility restriction can be changed by updating the Seldon Deploy helm chart environment variable `USERID_CLAIM_KEY`. This value defaults to `preferred_username` but can be configured to other user info claim values.

```
env:
  USERID_CLAIM_KEY:"name" # claim to be used as userid (defaults to "preferred_
↪username")
```

7.6.3 Using Open Policy Agent policies

With *OPA enabled* the visibility of namespaces can be controlled via authorization policies.

To understand how the authorization policies work in more detail please see *their documentation* first.

7.6.3.1 Global Visibility

Making namespaces globally visible is possible by giving `read` and `write` permissions to that namespace to all users (`*`). An example policy will look like this:

```
{
  "role_grants": {},
  "user_grants": {
    "*": [
      {
        "action": "read",
        "resource": "namespace/seldon"
      },
      {
        "action": "write",
        "resource": "namespace/seldon"
      }
    ]
  }
}
```

This policy file is a bare minimum which will allow **all** users to see the `seldon` namespace and operate on resources in that namespace.

7.6.3.2 Group Based Visibility Restrictions

To give access to a namespace on a group level, the `role_grants` policy list can be used. The group list in the request is obtained from the `groups` claim in the OIDC authentication token. An example of giving the `data-scientist` group access to the `prod` namespace looks like this:

```
{
  "role_grants": {
    "data-scientist": [
      {
        "action": "read",
        "resource": "namespace/prod"
      },
      {
        "action": "write",
        "resource": "namespace/prod"
      }
    ]
  },
  "user_grants": {}
}
```

7.6.3.3 User-Based Visibility Restrictions

To give access to a namespace on a user level, the `user_grants` policy list can be used.

The claim to be used for user-based visibility restriction can be changed by updating the Seldon Deploy helm chart environment variable `USERID_CLAIM_KEY`. This value defaults to `preferred_username` but can be configured to other user info claim values.

```
env:
  USERID_CLAIM_KEY:"name" # claim to be used as userid (defaults to "preferred_
↪username")
```

An example of giving the `alice` user access to the `prod` namespace looks like this:

```
{
  "role_grants": {},
  "user_grants": {
    "alice": [
      {
        "action": "read",
        "resource": "namespace/prod"
      },
      {
        "action": "write",
        "resource": "namespace/prod"
      }
    ]
  }
}
```

7.7 Namespace Setup

Setting up kubernetes namespaces

Namespaces can have various features. A basic configuration can be done first. This is enough to use many features. Further features such as gitops can be added later.

7.7.1 Basic Namespace Setup

7.7.1.1 With namespace labels

By default Seldon Deploy does not show any namespaces that are not explicitly labelled.

You can make a namespace globally available to all users by providing the following label to the namespace:

```
seldon.restricted: "false"
```

This is the minimum needed to use a namespace.

7.7.1.2 With authorization policies

With *OPA enabled* you can make namespaces globally visible by giving `read` and `write` permissions to that namespace to all users (`*`). An example policy will look like this:

```
{
  "role_grants": {},
  "user_grants": {
    "*": [
      {
        "action": "read",
        "resource": "namespace/seldon"
      },
      {
        "action": "write",
        "resource": "namespace/seldon"
      }
    ]
  }
}
```

This policy file is a bare minimum which will allow all users to see the `seldon` namespace.

7.7.2 Namespace Filtering

7.7.2.1 With namespace labels

If the label `seldon.restricted` is set to "true" then visibility of the namespace can be narrowed by group or by user.

For details of this, see *Namespace Visibility*

7.7.2.2 With authorization policies

With *OPA enabled* non-global namespaces are supported by both group and user based policies.

For details of this, see *Namespace Visibility*

7.7.3 Batch Jobs Permissions

Batch jobs use argo as a workflow engine. Argo needs a service account in the namespace and this has to be given permissions to create necessary resources. This is explained in the *Argo section*

7.7.4 Object Storage Secrets

The Seldon Deploy demos use models hosted in public buckets. To use private buckets, a secret containing environment variables should be setup in the namespace. An example is provided in the *Argo section*.

Models should reference the secret in the `envFrom` section of the wizard.

7.7.5 Gitops

A GitOps setup utilizes argocd to manage namespace contents in git. This requires setup for git, argocd and per-namespace. Each of these elements is covered in *the gitops section*

7.7.6 Multi-tenant Permissions (Only Used for Multi-tenant Installations)

In a multi-tenant setup, deploy is not automatically set up with permissions to work with multiple namespaces. Each namespace then needs further permissions. Details on this are in the *multitenant section*

7.7.7 Seldon-logs namespace (a special namespace)

The seldon-logs namespace is used for request logging. It is a special namespace and its setup is covered in *the request logging section*. It can be setup as a gitops namespace but this is optional. ML models do not go in this namespace.

7.8 Authorization Policies Management

Managing authorization policies when using Open Policy Agent authorization.

Open Policy Agent (OPA) policies is one of the ways of managing *authorization in Deploy*. To enable it follow the *production installation guide* or enable it via the trial install setup.

7.8.1 Policy schema

The OPA policies are a list of resource-action pairs, defined in a JSON files that must be present to Seldon Deploy on startup. The Helm chart is able to load the file from a config map specified in the *installation guide*, but any other way of providing the file will also work.

7.8.1.1 Targets

Seldon Deploy can authorize users based on their **groups** and their **ID**. This information is obtained from the OIDC token fetched during the *authentication step*.

The claim to be used for user-based authorization can be changed by updating the Seldon Deploy Helm chart environment variable `USERID_CLAIM_KEY`. This value defaults to `preferred_username` but can be configured to other user info claim values.

```
env:
  USERID_CLAIM_KEY:"name" # claim to be used as userid (defaults to "preferred_
↪username")
```

The base schema of the policy file consists of two fields - `role_grants` and `user_grants` mapping the groups and user IDs respectively:

```
{
  "role_grants": {
//    policies based on the user's groups...
  },
  "user_grants": {
//    policies based on the user's ID...
  }
}
```

A special target is the user `*` which can be used to give global policies. To see how this can be used check the *examples section*.

7.8.1.2 Resources

Seldon Deploy serves various resources like ML models and the deployments of these models. Right now these resources are grouped in two categories:

- Namespace based resources - deployments.
- Project based resources - models.

The actions one can perform on these resources are `write` and `read`, where `write` corresponds to all actions that modify a resource like crating it, updating it, and deleting it. `read` corresponds to actions that only inspect the resource but do not modify it.

A policy resource-action pair looks like this:

```
{
  "action": "read",
  "resource": "namespace/seldon"
}
```

For more details on the specifics of each resource type policies please see the *namespace visibility* and *project visibility* guides.

7.8.1.3 Examples

A full example of a policy file looks like this:

```
{
  "role_grants": {
    "data_scientist": [
      {
        "action": "read",
        "resource": "project/iris"
      },
      {
        "action": "write",
        "resource": "project/iris"
      },
      {
        "action": "read",
        "resource": "namespace/seldon"
      }
    ],
    "ops": [
      {
        "action": "read",
        "resource": "project/iris"
      },
      {
        "action": "read",
        "resource": "namespace/seldon"
      },
      {
        "action": "write",
        "resource": "namespace/seldon"
      }
    ]
  },
  "user_grants": {
    "*": [
      {
        "action": "read",
        "resource": "project/default"
      },
      {
        "action": "write",
        "resource": "project/default"
      },
      {
        "action": "read",
        "resource": "namespace/default"
      },
      {
        "action": "write",
        "resource": "namespace/default"
      }
    ],
    "alice": [
      {
        "action": "read",
        "resource": "namespace/alice"
```

(continues on next page)

(continued from previous page)

```
      },
      {
        "action": "write",
        "resource": "namespace/alice"
      }
    ]
  }
}
```

In this example, users part of the `data_scientist` group will be able to create, modify, and read models from the `iris` project, as well as see deployments in the `seldon` namespace. However, they do not have permissions to deploy or modify deployments in that namespace. Similarly, users part of the `ops` group do not have permissions to modify models in the `iris` project, but can see these models as well as deploy to the `seldon` namespace.

The `*` user grant specifies that all users have read and write permissions on the `default` project and namespace. Another example of the user grant is the permissions given to the user `alice` - no matter their groups, they will have read and write access to the `alice` namespace.

7.8.2 Policy evaluation

The final list of policies that a user has is a combination of:

- the policies that they explicitly have from the `user_grants` policy list.
- the policies given to all users - the policies in the `user_grants` list for the `*` user.
- all the policies in the `role_grants` list associated with groups that the user belongs to.

Right now, negative authorization (denials) is not supported.

7.8.3 Policy operations

The policy file can be modified and stored in various ways. The simplest way to start using OPA policies in Seldon Deploy is by mounting it from a config map as described in the *production installation guide*.

To interact with the policy config map there is a utility command line tool that can be downloaded from the seldon-deploy-resources repo. Once downloaded the correct binary for your architecture you can start using the `policy_client` tool. For full documentation of the tool you can run `policy_client --help`.

Seldon Deploy listens for changes to the policy file and will automatically reload the policies if they change.

7.8.3.1 Migrating from Namespace labels

To migrate from using namespace labels for defining access requirements to a namespace you can use the following command:

```
policy_client sync
```

It will create a config map called `seldon-deploy-policies` in the `seldon-system` namespace if one is not present. The name and namespace are configurable by flags. It will then populate policies corresponding to the namespace labels in the cluster. The command will not delete any existing policies in the config map.

To confirm the change has been successful you can check the config map by running:

```
kubectl get cm -nseldon-system seldon-deploy-policies -o jsonpath='{.data.data}' | jq
```

7.8.3.2 Adding a new policy

To add a new policy you can run:

```
policy_client addPolicy --resource=namespace/seldon --action=write --target_
↪users=alice --target_groups=admin
```

The arguments `target_groups` and `target_users` can take multiple entries.

7.8.3.3 Deleting an existing policy

Similarly to adding, to remove an existing policy you can run:

```
policy_client removePolicy --resource=namespace/seldon --action=write --target_
↪users=alice --target_groups=admin
```

7.8.3.4 Limitation

Using a config map for the policy file is flexible and easy to setup. However, it comes with a limitation which is true for all Kubernetes resources - it is limited to 1MB size since the underlying etcd value limit. If your use case requires a bigger size for the policy file consider mounting the file from somewhere else like an object store (S3, minio), or a database. We are working on adding native support for such use cases which should be generally available in future releases.

7.9 Visibility of projects

Seldon Deploy allows models to be grouped into projects. A project can have multiple models associated with it. It is possible to make authorization decisions based on the project a model is associated with. This guide details how this can be done.

7.9.1 Setup

To start using project based authorization, it must be enabled together with OPA auth as per the *installation instructions*. To see how the OPA policies for project based resources look like go to the *policy section*.

7.9.2 Protected resources and policy setup

7.9.2.1 Models

Right now models are the only resources that are explicitly associated with a project. By default, all models that do not specify a project are assigned to the `default` project.

It is strongly advised to set up a base policy so that these models are visible to everyone. A section like this in the policy file will achieve this:

```
{
  "role_grants": {},
  "user_grants": {
    "*": [
      {
        "action": "read",
        "resource": "project/default"
      },
      {
        "action": "write",
        "resource": "project/default"
      }
    ]
  }
}
```

If a user has a `read` permission on a project it means they can read all models associated with that project. However, they won't be able to perform actions like updating the model, creating a new model in that project, or deleting a model from that project, unless they have a `write` permission on the project as well. These authorization checks are performed both if interacting directly with the Seldon API, or via the Model Catalog in the UI.

7.9.2.2 Deployments

Since Seldon Deployments and Inference Services reference models themselves, these resources are also protected by project based policies. It is possible for a deployment to consist of multiple models that can belong to different projects. A user must have a `read` permission on **all** models referenced in a Seldon Deployment or Inference Service in order to see it. *Namespace level authorization* is still applied.

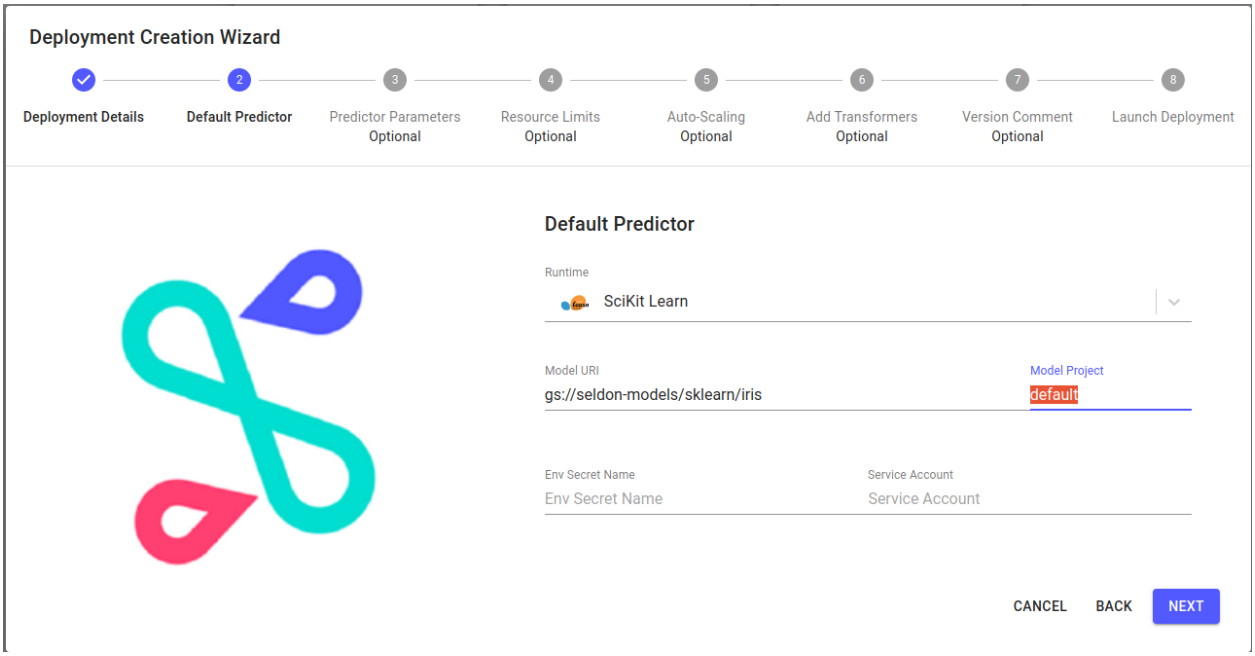
To see how Seldon Deployment and Inference Services reference which project a model they reference belongs to see the *next section*.

7.9.3 Associating resources in Kubernetes with a project

Neither Seldon Deployments nor Inference Services support the concept of projects natively. Therefore, we are using annotations on the CRDs to link a model in that resource with its corresponding project.

7.9.3.1 When deploying through the UI

When modifying Seldon Deployment or an Inference Service through the Deploy UI it will automatically be annotated with the projects that are referenced in the `Model Project` field. If none is specified, the `default` project will be assumed.



7.9.3.2 When deploying manually (gitops, kubectl, etc.)

If you are manually constructing the yaml/json that you want to be deployed to the Seldon Deploy cluster, you must annotate the deployments for Deploy to know what project a model refers to. If no annotations are present, the `default` project will be assumed. Here is how to annotate a deployment:

Seldon Deployment

Seldon Deployments' models are annotated on the predictor level. Each predictor in the `spec.predictors` field has an `annotations` field that contains annotations like `"project.seldon.io/$name_of_component_in_graph": "$project_name""`. This allows us to specify the models of all components in a predictor's deployment graph since the projects can be different.

Here is a simplified example of the spec of a Seldon Deployment with only one single-model predictor:

```
{
  "spec": {
    "name": "mock-example",
    "predictors": [
      {
        "name": "default",
        "graph": {
          "name": "mock-example-container",
          "type": "MODEL",
          "logger": {
            "mode": "all"
```

(continues on next page)

(continued from previous page)

```
          }
        },
        "componentSpecs": [
          {
            "spec": {
              "containers": [
                {
                  "name": "mock-example-container",
                  "image": "seldonio/mock_classifier:1.5.0",
                  "resources": {
                    "limits": {
                      "cpu": "1",
                      "memory": "1Gi",
                      "nvidia.com/gpu": "0"
                    },
                    "requests": {
                      "cpu": "100m",
                      "memory": "1Gi",
                      "nvidia.com/gpu": "0"
                    }
                  }
                }
              ]
            }
          }
        ],
        "replicas": 1,
        "annotations": {
          "project.seldon.io/mock-example-container": "iris"
        },
        "engineResources": {},
        "svcOrchSpec": {},
        "traffic": 100
      }
    ],
    "annotations": {
      "seldon.io/engine-seldon-log-messages-externally": "true"
    },
    "protocol": "seldon"
  }
}
```

The only part different from the default Seldon Core spec is the `annotations` field specifying the project of the model in `mock-example-container` node.

Inference Service

Inference Services do not have predictor level annotations, so we use the annotations on the Inference Service level instead. Again, we follow the same pattern as with Seldon Deployments for the annotations. They look like `"project.seldon.io/$name_of_component": "$project_name"`. However, the component names are predefined here. The available annotation options are:

- `"project.seldon.io/default": "$project_name"` - the default predictor
- `"project.seldon.io/default-transformer": "$project_name"` - the transformer of the default predictor

- `"project.seldon.io/canary": "$project_name""` - the canary predictor
- `"project.seldon.io/canary-transformer": "$project_name""` - the transformer of the canary predictor

A complete example of an inference service with canary and transformers for both the canary and the default predictors:

```
{
        "kind": "InferenceService",
        "apiVersion": "serving.kubeflow.org/v1alpha2",
        "metadata": {
                "name": "mock-example",
                "namespace": "seldon",
                "resourceVersion": "7249641",
                "labels": {
                        "fluentd": "true"
                },
                "annotations": {
                        "project.seldon.io/default": "iris",
                        "project.seldon.io/default-transformer": "income",
                        "project.seldon.io/canary": "default",
                        "project.seldon.io/canary-transformer": "income"
                }
        },
        "spec": {
                "default": {
                        "predictor": {
                                "custom": {
                                        "container": {
                                                "name": "",
                                                "image": "seldonio/mock_classifier:1.
↪5.0",
                                                "resources": {
                                                        "limits": {
                                                                "cpu": "1",
                                                                "memory": "2Gi"
                                                        },
                                                        "requests": {
                                                                "cpu": "1",
                                                                "memory": "2Gi"
                                                        }
                                                }
                                        }
                                },
                                "logger": {
                                        "mode": "all"
                                }
                        },
                        "transformer": {
                                "custom": {
                                        "container": {
                                                "name": "transformer-container",
                                                "image": "seldonio/mock_classifier:1.
↪5.0",
                                                "resources": {
                                                        "limits": {
                                                                "cpu": "1",
                                                                "memory": "2Gi"
```

(continues on next page)

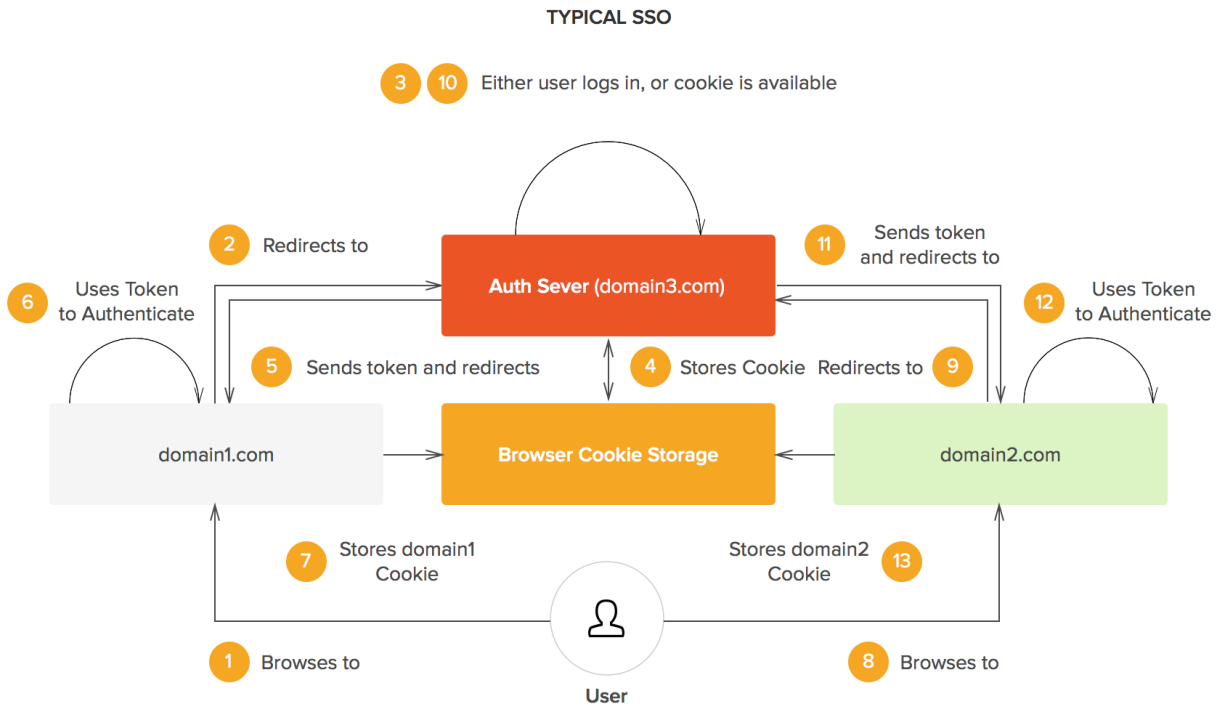
(continued from previous page)


```
                                                        },
                                                        "requests": {
                                                                "cpu": "1",
                                                                "memory": "2Gi"
                                                        }
                                                }
                                        }
                                }
                        },
                        "canary": {
                                "predictor": {
                                        "custom": {
                                                "container": {
                                                        "image": "seldonio/mock_classifier:1.
↪5.0",
                                                        "env": []
                                                },
                                                "resources": {
                                                        "requests": {
                                                                "cpu": "0.1",
                                                                "memory": "1Gi",
                                                                "nvidia.com/gpu": "0"
                                                        },
                                                        "limits": {
                                                                "cpu": "1",
                                                                "memory": "1Gi",
                                                                "nvidia.com/gpu": "0"
                                                        }
                                                }
                                        },
                                        "logger": {
                                                "mode": "all"
                                        }
                                },
                                "transformer": {
                                        "custom": {
                                                "container": {
                                                        "name": "transformer-container",
                                                        "image": "seldonio/mock_classifier:1.
↪5.0"
                                                }
                                        }
                                }
                        },
                        "canaryTrafficPercent": 10
                },
                "status": {}
}
```

7.10 Single sign-on (SSO)

Single sign-on (SSO) is a property of identity and access management (IAM) that enables users to securely authenticate with multiple applications by logging in only once (with just one set of credentials). For SSO, Seldon Deploy relies on the OIDC protocol that handles authentication through JSON Web Tokens and a central identity provider to verify user identity.

A typical enterprise SSO flow for web based applications is as follows



As shown in the diagram, Seldon Deploy sso integration works if any enterprise application on `domain1.com` and a Seldon Deploy installation on `domain2.com` is connected to a common auth Server on `domain3.com`.

Currently, Seldon Deploy SSO can be configured in two ways.

- kubeflow cluster
- *app-level authentication*

7.11 Theme Customization

Seldon Deploy allows customization of the user-interface theme. Seldon Deploy follows Material Design for building its user experience And most of the theming and design aspects are based on a popular Material design library called Material UI.

Seldon Deploy supports complete custom configuration the material theme and the Logo image in the UI header.

7.11.1 Customization

1. To get started create a new folder named `custom`.
2. Create a `config.json` file in this folder.
3. Add two sections to the `config.json` file namely `logo` and `theme`.

```
{
    "logo":"",
    "theme":{}
}
```

1. The logo section is a sting that represents filename of the new logo Image. Do remember to add the image file , in this case `custom_logo.png`, to the `custom` folder as well.
2. The theme section is a material-ui theme configuration used to customize the entire UI as needed. For further options for configuration and customization, refer material ui theming and default theme
3. So a basic theme configuration would look like as follows,

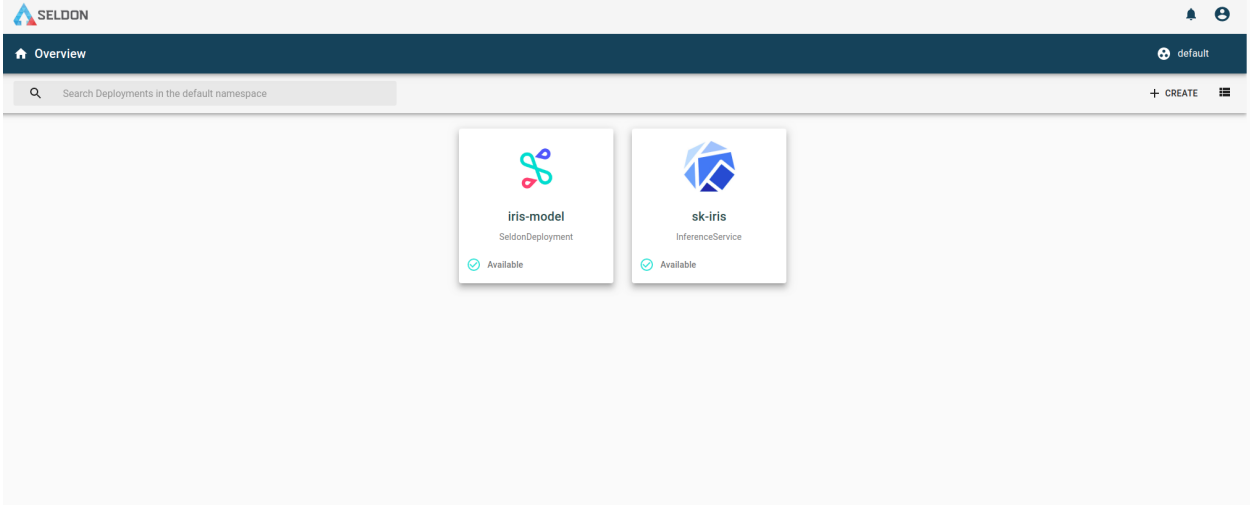
```
{
  "logo": "/custom_logo.png",
  "theme": {"palette": {"primary": {"main": "#15425a"}}}
}
```

7.11.2 Apply custom theme configuration

1. Now that the `custom` folder is ready, create a kubernetes secret named `seldon-deploy-custom-theme` in the `seldon-system` namespace.

```
kubectl create secret -n seldon-system generic seldon-deploy-custom-theme --from-
↪file=./custom
```

1. The Seldon Deploy pods will automatically use this custom configuration and load appropriately.



7.11.3 Remove custom theme configuration

1. Remove the secret to revert back to default configuration

```
kubectl delete secret -n seldon-system seldon-deploy-custom-theme
```

7.12 Usage Monitoring

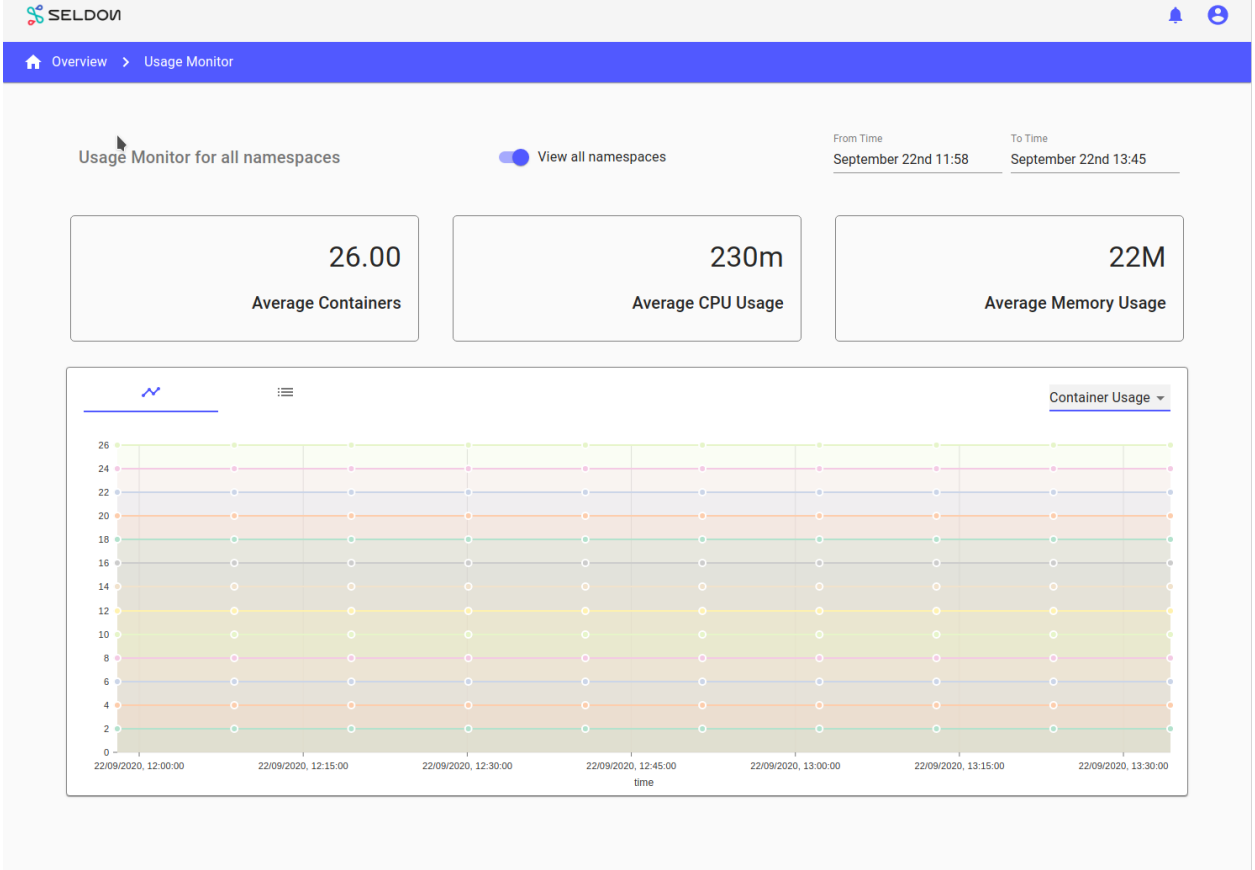
Summary metrics are available showing usage over time. These can be accessed from 'Usage' on the top-right menu.

These metrics differ from those shown for individual models on their pages. The summary metrics show multiple models for a namespace or cluster.

By default if the cluster-level is chosen then all models are counted for all namespaces. If you wish to restrict to namespaces visible to the user, set the FILTER_USAGE_MON_NAMESPACES environment variable to true.

Resources are shown broken down by model. The type of resource of interest can be chosen. The containers resource is shown with a comparison against what your license allows for.

Models are stacked on top of each other in the breakdown when there are multiple


SELDON
Overview
Usage Monitor
Usage Monitor for all namespaces
View all namespaces
From Time
September 22nd 11:58
To Time
September 22nd 13:45
26.00
Average Containers
230m
Average CPU Usage
22M
Average Memory Usage
Container Usage
26
24
22
20
18
16
14
12
10
8
6
4
2
0
22/09/2020, 12:00:00
22/09/2020, 12:15:00
22/09/2020, 12:30:00
22/09/2020, 12:45:00
22/09/2020, 13:00:00
22/09/2020, 13:15:00
22/09/2020, 13:30:00
time

CHAPTER

EIGHT

UPGRADING

Upgrading Seldon Deploy

8.1 Upgrading to 1.3.0

- Seldon Core v1.8.0 changed the default storage initializer to be `seldonio/rclone-storage-initializer`. The secret format that holds credentials have changed and requires update.
  - If you do not wish to update them now, please, set following value in `core-values.yaml` used when installing Seldon Core with Helm

```
storageInitializer:
  image: gcr.io/kfserving/storage-initializer:v0.4.0
```

  This is documented in kfserving storage initializer documentation.
  - Following resources are available to help you upgrade to the rclone-compatible secret format:
    * Seldon Core 1.8 upgrading documentation
    * Handling Credentials documentation for pre-packaged model servers with rclone-based storage initializer
    * Tutorial on how to test new secret format without setting rclone-based storage initializer globally
    * An example upgrade path that illustrates steps required to smoothly upgrade AWS S3 / MinIO configured clusters from kfserving-based `Secret` and `Service Account` credentials to the new format.
- SD 1.3.0 introduces improvement to the Batch Processing of requests - generated Argo workflows now use storage initializer mechanism for pulling in input file and uploading output file. The default storage initializer is `seldonio/rclone-storage-initializer` which requires adjustment of credential secret.
  - If you do want to keep using current format of secret without updating it now, set following value in `deploy-values.yaml` used when installing Seldon Deploy with Helm

```
batchjobs:
  storageInitializer:
    image: gcr.io/kfserving/storage-initializer:v0.4.0
```

  - To upgrade to the new format see links provided in the previous point. Also consult our documentation that contains example for MinIO-compatible secret.
- SD 1.3.0 updates the request logger component's image in the helm chart default values file to `seldonio/seldon-request-logger:1.9.1`. This version of request logger looks up model level metadata from Seldon Deploy Metadata service before creating predictions logging elasticsearch indexes. This feature is

leveraged by new advanced monitoring features like *distributions monitoring* introduced in this SD version. Although, it is not a mandatory configuration and it is possible disable the metadata lookup from the request logger. New features like distributions monitoring will still work, you just won't be able to enrich logged predictions with the model level predictions schema (e.g. feature category names).

  - As mentioned earlier, metadata lookup is not mandatory for the request logger. If you wish to disable this feature and would prefer the request logger to create the elasticsearch indexes without the model level metadata as in previous releases, follow the steps below.
    * If you have already enabled model level metadata feature from version v1.2.0 and want to continue using this feature without the request logger connection to Seldon Deploy, then use the default helm chart values in the request logger section
    * Alternatively, if you have disabled model level metadata feature by setting the `metadata.pg.enabled` to false also disables the request logger connection to Seldon Deploy as the model level metadata is not being stored.
  - To enable this metadata lookup feature, the request logger requires authentication configuration to connect with a Seldon Deploy instance. For configuring this feature, see the request logger page for details of the authentication setup needed to connect to Seldon Deploy. If you are using model metadata then adding auth from the request logger to Deploy enables the prediction schema for a model to be found and used to enrich prediction data and enrich monitoring. This will be enabled for all new elasticsearch logging indexes once setup. Existing indexes will work as before but will need to be reindexed to store additional enriched info. Enabling auth from request logger to Deploy currently requires a client credentials auth flow but can also be configured to work with the password grant flow if needed.

8.2 Upgrading to 1.2.0

- 1.2.0 introduces the *Model Metadata storage* to Seldon Deploy. To enable/disable the model metadata storage functionality with all required prerequisites follow the *instructions*. The quick overview of the changes is:
  - If you do not want the model metadata functionality disable it by setting `metadata.pg.enabled` in the Seldon Deploy helm chart to `false`. Not recommended since all model metadata related features will be unavailable.
  - Install a self-hosted or managed PostgreSQL solution (instructions).
  - Configure the connection details to the PostgreSQL instance by adding a `metadata-postgres` secret in the Seldon Deploy namespace (instructions).
  - Enable the model metadata storage by setting `metadata.pg.enabled` in the Seldon Deploy helm chart to `true` (instructions).
- Detector components for outlier detection, drift detection and metrics are now moving out of the `seldon-logs` namespace and into the user namespaces. To handle this
  - The easiest way to migrate is to delete any existing detectors from the UI and recreate them after upgrade.
  - To see what detectors you have, do `kubectl get ksvc -n seldon-logs`. Also check `kubectl get trigger -n seldon-logs`.
  - Detectors will have `-outlier`, `-drift` or `-metrics` in their names. Each will have a `ksvc` and a corresponding `trigger`. The name and namespace of the model they are for will also be in the name.
  - If you find any issues while carrying out this process, please contact Seldon.
- The *ElasticSearch integration* will now enable basic authentication by default.
  - This is controlled by the `elasticsearch.basicAuth` flag in the Seldon Deploy Helm chart.

  - To disable it (and revert to the default behaviour of `<= 1.1.0`), you can set its value as `elasticsearch.basicAuth=false`.
  - To keep it enabled and configure ElasticSearch's user / password credentials, please check the *ElasticSearch integration setup guide*.
- The Seldon Core version is now `1.9.1`. See the `Getting Started > Production Installation` section for installation. A `helm upgrade --install` can be used to upgrade.
- The demo drift detector has changed from `gs://seldon-models/alibi-detect/cd/ks/cifar10-0_4_3` to `gs://seldon-models/alibi-detect/cd/ks/cifar10-0_4_4`. This is because the alibi version in Seldon Core is upgraded.

8.3 Upgrading to 1.1.0

- The helm values file now has an option to set user and password for basic auth to Elastic. For more information, see the *request logger documentation*.
- The recommended prometheus settings have now changed to improve performance. We suggest uninstalling the prometheus helm chart and *installing* with the below changes.
  - See the *metrics page* for the full helm values.
  - The parts that have changed are:
    * `prometheus.server.livenessProbePeriodSeconds` is now 30
    * `prometheus.server.extraArgs` has been added with entries `query.max-concurrency: 400` and `storage.remote.read-concurrent-limit: 30`
    * `prometheus.server.resources.limits` now has `cpu: 2` and `memory: 4Gi`
    * `prometheus.server.resources.requests` now has `cpu: 800m` and `memory: 1Gi`
  - Remove analytics with `helm delete -n seldon-system seldon-core-analytics`.
  - Install again using the *instructions*
- The recommended Seldon Deploy helm values have changed slightly to improve performance. Set the following:
  - `resources.limits` should now have `cpu: 800m` and `memory: 800Mi`
- A further change to the Seldon Deploy helm values is the addition of a configuration for the runAsUser for knative pods started by Deploy:

```
defaultUserID: "8888"
```

8.4 Upgrading to 1.0.0

- If you are upgrading by more than one version, please see previous notes.
- The self-install trial customers, the file `dockercreds.txt` is now called `sdconfig.txt` (anyone using `Product Installation` can ignore this).
  - The variable `KUBEFLOW_USER_EMAIL` has become `SD_USER_EMAIL` and `KUBEFLOW_PASSWORD` has become `SD_PASSWORD`
- New components have been introduced for Batch support. Argo and minio. See `Getting Started > Production Installation`

  - Note the new Batch component requires per-namespace config for any Batch-enabled namespaces. See `Getting Started > Production Installation` section on `Argo` and refer to references to .

- For upgrading Seldon Deploy specifically
  - The `requestLogger.image` in the helm values file is now `docker.io/seldonio/seldon-request-logger:1.5.0`
  - The `alibidetect.image` in the helm values is now `seldonio/alibi-detect-server:1.5.0`.
  - There's a new entry in the `batchjobs` section of the helm values - `serviceAccount: "workflow"`
  - The Seldon Core version is now 1.5.0. See the `Getting Started > Production Installation` section for installation.
  - The KFServing version is 0.4.1.

8.5 Upgrading to 0.9.0

- Seldon Deploy 0.9.0 uses knative 0.18. To upgrade this, see the *Upgrading Knative* section.
- For upgrading Seldon Deploy specifically
  - A license is required at app install/startup time. Contact Seldon for a license.
  - Seldon Core Analytics or Prometheus needs to be upgraded. Prometheus recording rules have been introduced. See `Getting Started > Production Installation > Metrics Monitoring`
  - If Seldon Core Analytics is used, upgrade with using `seldon-deploy-install/prerequisites-setup/prometheus/seldon-core-analytics.sh` from the scripts extracted in `Getting Started > Download Installation Resources`.
  - Upgrading Seldon Core Analytics will reset prometheus data.
  - The `requestLogger.image` in the helm values file is now `docker.io/seldonio/seldon-request-logger:1.3.0-json`
  - The `requestLogger.trigger` in the helm values now has `apiVersion` "eventing.knative.dev/v1" and `broker` "default".
  - The `alibidetect.image` in the helm values is now `seldonio/alibi-detect-server:1.4.0`.
  - There is a new `batchjobs` section in the helm values.

8.6 Upgrading to 0.8.2

- In installations that have restricted external access (eg. air gapped), the following image will need to be downloaded:
  - docker.io/seldonio/seldon-request-logger:1.3.0-json

8.7 Upgrading to 0.8.0

- The sd-install script is now called sd-install-default. Or for a kubeflow install there's sd-install-kubeflow.
- The helm values file now has a single gitops section. This involves the following changes.
  - The recommended way to configure git credentials is in a git secret, though a token parameter to the chart can still be used.
  - The sd-install-default script contains an example of this. It is:

```
kubectl create secret generic git-creds -n seldon-system --from-file ~/.ssh/id_rsa --
↪from-file ~/.ssh/known_hosts --from-literal=passphrase="$GIT_SSHKEY_PASSPHRASE" --
↪from-literal=username="$GIT_USER" --from-literal=token="$GIT_TOKEN" --from-
↪literal=email="$GIT_EMAIL"  --dry-run -o yaml | kubectl apply -f -
```

- skipVerifyGit, gitwebhook, GITOPS_FORMAT and argocd config have all moved from `github` to the new `gitops` section in helm values file, see documentation.
- if your ArgoCD Application does not follow naming pattern `seldon-gitops-${namespace}`, please, specify the ArgoCD application using namespace labels, see documentation.
- The contents of the helm chart have also changed. In particular:
  - The istio virtual service no longer does path rewriting.
  - Permission is now required to watch argocd applications.
  - The liveness and readiness probes are now on `/seldon-deploy/api/status`

8.8 Upgrading to 0.7.0

- The Deploy helm chart now contains an option to install a request logger in the seldon-logs namespace. This is on by default. Previously loggers were per-namespace. To use a single request logger installed through the chart, ensure the seldon-logs namespace is labelled for knative-evenint and that seldon core is v1.2.1 and has executor.requestLogger.defaultEndpoint http://default-broker.seldon-logs. For KfServing, if using it, the default broker will for now continue to be in the current namespace unless set at the point of creating the model deployment.
- The helm values file has seen some changes.
  - There's now a request logger section for installing the request logger to a dedicated namespace, if desired.
  - The istioIngress section has been renamed ingressGateway
  - There are extra options in the prometheus section for for using a secured prometheus (assumes not secure by default, as before).
  - There are extra options in the elasticsearch section for for using a secured prometheus (assumes not secure by default, as before).

8.8.1 Upgrading Knative

How to upgrade knative for Seldon Deploy

8.8.1.1 Upgrading to 0.18

Most likely you are currently running 0.11 of knative. We suggest removing and installing a new version.

The official knative docs suggest upgrading versions one by one.

We suggest deleting the existing install and installing a new one in order to save time.

Note that knative serving 0.18 officially requires at least istio 1.4.10 but has been known to work with 1.2.10.

If you do not meet the minimum istio version, consider only upgrading knative eventing. Seldon Deploy can work with knative serving 0.11. To only upgrade eventing, modify the steps below to remove anything related to serving.

Note that we will delete all existing Triggers and kservices. But if these are just for Seldon Deploy then they can be recreated.

If you have InferenceServices running then there will be downtime for these resources (their kservices will be removed and not recreated until patching KFServing). Any outlier detectors should be deleted and recreated afterwards.

First remove Deploy with `helm delete -n seldon-system seldon-deploy`

To remove knative 0.11 run:

```
#remove all knative eventing setup from seldon-logs namespace
kubectl label namespace seldon-logs knative-eventing-injection-
kubectl delete -n seldon-logs trigger.eventing.knative.dev/seldon-request-logger-
↪trigger
kubectl delete broker -n seldon-logs default
kubectl delete -n seldon-logs inmemorychannel.messaging.knative.dev/default-kne-
↪ingress
kubectl delete -n seldon-logs inmemorychannel.messaging.knative.dev/default-kne-
↪trigger
kubectl delete -n seldon-logs -l eventing.knative.dev/broker=default
kubectl delete -n seldon-logs subscription.messaging.knative.dev/internal-ingress-
↪default-db21c71f-6a5f-4216-81b2-bcedfba7d535
kubectl delete -n seldon-logs service.serving.knative.dev/seldon-request-logger

#eventing webhooks
kubectl delete mutatingwebhookconfigurations.admissionregistration.k8s.io␣
↪sinkbindings.webhook.sources.knative.dev
kubectl delete mutatingwebhookconfigurations.admissionregistration.k8s.io webhook.
↪eventing.knative.dev
kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io␣
↪sinkbindings.webhook.sources.knative.dev
kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io config.
↪webhook.eventing.knative.dev
kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io␣
↪validation.webhook.eventing.knative.dev

#serving webhooks
kubectl delete mutatingwebhookconfigurations.admissionregistration.k8s.io webhook.
↪serving.knative.dev
kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io config.
↪webhook.serving.knative.dev
kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io␣
↪validation.webhook.serving.knative.dev
```

(continues on next page)

(continued from previous page)

```
kubectl delete --selector knative.dev/crd-install=true \
        --filename https://github.com/knative/serving/releases/download/v0.11.1/
↪serving.yaml \
        --filename https://github.com/knative/eventing/releases/download/v0.11.0/
↪eventing.yaml \
        --filename https://github.com/knative/serving/releases/download/v0.11.1/
↪monitoring.yaml

kubectl delete \
        --filename https://github.com/knative/serving/releases/download/v0.11.1/
↪serving.yaml \
        --filename https://github.com/knative/eventing/releases/download/v0.11.0/
↪eventing.yaml \
        --filename https://github.com/knative/serving/releases/download/v0.11.1/
↪monitoring.yaml

kubectl delete \
    ---filename https://github.com/knative/eventing/releases/download/v0.11.0/in-
↪memory-channel.yaml

kubectl delete namespace knative-eventing
kubectl delete namespace knative-serving
kubectl delete namespace knative-monitoring

#ensure knative eventing crds are gone
kubectl delete crd inmemorychannels.messaging.knative.dev
kubectl delete crd brokers.eventing.knative.dev
kubectl delete crd triggers.eventing.knative.dev
```

Next install knative serving and eventing. We suggest using `seldon-deploy-install/prerequisites-setup/knative/install_knative.sh` from the scripts extracted in `Getting Started > Download Installation Resources`.

A knative broker has to be created in the `seldon-logs` namespace. We suggest using `seldon-deploy-install/prerequisites-setup/efk/eventing_for_logs_ns.sh`.

Seldon Core and KFServing both need to be patched to point to the new form of knative broker url. We suggest using `seldon-deploy-install/prerequisites-setup/seldon-core/install_seldon_core.sh` and `seldon-deploy-install/prerequisites-setup/kfserving/install_kfserving.sh`

Seldon Deploy itself can then be upgraded. This will create a new instance of the request logger and its Trigger.

CHAPTER

NINE

HELP AND SUPPORT

Get help and support for Seldon Deploy

Obtain support to use Seldon Deploy for your enterprise by getting in touch with Seldon.

9.1 Troubleshooting

Seldon Deploy installs into standard kubernetes clusters and can be accessed by all modern browsers mentioned in the specifications.

Contact the Seldon team with the relevant issue and provide necessary details about your particular install of Seldon Deploy. The version details can be found at the about page of the tool that can be accessed from the user menu on the top-right corner.

This about page displays the Seldon Deploy version details, license details and your browser version. For more details on your specific browser, please visit https://www.whatsmybrowser.org/ and share the URL that can provide us with more information about your browser usage like resolution and other support needed.

9.1.1 Browser Specifications

We recommend using the latest browser version available to your operating system. See your browser's documentation to learn more about checking and updating your version. The project supports all modern browsers based on global usage (> 0.2%). The full list of browser versions supported include stable versions of Chrome, Firefox, Safari, Microsoft Edge and has a global coverage of 91.09%. The full set is listed here

```
and_chr 86
and_ff 82
and_uc 12.12
chrome 86
chrome 85
chrome 84
chrome 83
chrome 80
chrome 49
edge 86
edge 85
edge 18
firefox 82
```

(continues on next page)

(continued from previous page)

```
firefox 81
ios_saf 14
ios_saf 13.4-13.7
ios_saf 13.3
ios_saf 12.2-12.4
ios_saf 10.3
ios_saf 9.3
opera 71
safari 14
safari 13.1
safari 13
samsung 12.0
```

Important browser settings To log into and edit your site, check if these browser settings are enabled or disabled in your browser:

- Cookies - Cookies must be enabled in your browser, per our Cookie Policy. Blocking cookies will interfere with editing your site. JavaScript - JavaScript must be enabled to edit your site. Avoid programs that block JavaScript, like anti-virus software.
- Browser add-ons or extensions - Browser add-ons might interfere with site editing. While disabling them isn't always required, we may ask you to disable them when helping you troubleshoot.
- Browser window sizes Your computer's screen size determines the maximum browser window resolution. For the best experience editing your site, use a browser window at least 1280 pixels wide and 768 pixels tall.

9.1.2 Errors from Seldon Deploy

If Seldon Deploy crashes or returns an error the best first steps to get details are:

1. Turn on the network tab in the browser (via right-click and 'Inspect' in chrome), hit 'Preserve Log' and recreate the issue. Then find the failed call to see the full message.
2. Find the Seldon Deploy pod (usually in the seldon-system namespace) and inspect its logs.

This should help determine if the cause is Seldon Deploy or another component. If another component, see the other debugging sections from here. Either way the information obtained will be useful if an issue needs to be reported to Seldon.

9.1.3 Insufficient ephemeral storage in EKS clusters

When using `eksctl`, the volume size for each node will be of 20Gb by default. However, with large images this may not be enough. This is discussed at length on this thread in the `eksctl` repository.

When this happens, pods usually start to get evicted. If you run `kubectl describe` on any of these pods, you should be able to see errors about `not enough ephemeral storage`. You should also be able to see some `DiskPressure` events on the output of `kubectl describe nodes`.

To fix it, it should be enough to increase the available space. With `eksctl`, you can do so by tweaking the `nodeGroups` config and adding a `volumeSize` and `volumeType` keys. For instance, to change the volume to 100Gb you could do the following in your `ClusterConfig` spec:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

...
```

(continues on next page)

(continued from previous page)

```
nodeGroups:
  - volumeSize: 100
    volumeType: gp2
    ...
```

9.1.4 Elastic Queue Capacity

If request logging is used with a high throughput then it's possible to hit a `rejected execution of processing` error in the logger. This comes with a `queue capacity` message. To address this the `thread_pool.write.queue_size` needs to be increased. For example, with the elastic helm chart this could be:

```
esConfig:
  elasticsearch.yml: |
    thread_pool.write.queue_size: 2000
```

9.1.5 Request Logs Entries Missing

Sometimes requests fail to appear in the request logs. Often this is a problem with the request logger setup. If so see the *request logging docs*.

Sometimes we see this error in the request logger logs:

```
RequestError(400, 'illegal_argument_exception', 'mapper [request.instance] cannot be␣
↪changed from type [long] to [float]')
```

What happens here is elastic has inferred the type of the fields in the request for the model's index. This is inferred on the first request and if it changes or is inferred incorrectly this has to be addressed manually.

The best thing to do here is to delete the index.

First port-forward elastic. If the example elastic is used then this is `kubectl port-forward -n seldon-logs svc/elasticsearch-opendistro-es-client-service 9200`.

To delete the index we need to know its name. These follow a pattern. The pattern is:

`inference-log-<seldon/kfserving>-<namespace>-<modelname>-<endpoint>`

Usually `endpoint` is `default` unless there's a canary.

You can also see indexes in kibana (in a trial install on /kibana/) or using the elastic API

Then delete the index with a curl. If the auth is `admin/admin` and there's a `cifar10` seldon model in a namespace also called `seldon` then it's `curl -k -v -XDELETE https://admin:admin@localhost:9200/inference-log-seldon-seldon-cifar10-default`

9.1.6 Request Logs Entries Incorrect

Model names should not overlap as at the time of writing there is a bug with the request log page. Its searches can get confused between, say, 'classifier' and 'classifier-one'. The logs in the backend are distinct but the searches can get muddled. This issue is being tracked - for now the workaround is to use distinct names. This affects both viewing the prediction logs and also the feature distributions monitoring feature. In the distributions monitoring dashboard, this error may manifest as missing statistics and distribution graphs.

9.1.7 Auth

See the *auth section* for debugging tips.

9.1.8 Knative

See the *knative install section* for how verify knative.

9.1.9 Argo and Batch

See the *argo section* for debugging batch and the *minio section* for minio.

9.1.10 Prometheus

See the *metrics section* for debugging prometheus.

9.1.11 Serving Engines

For Seldon Core and for KFServing debugging, it is best to see their respective docs.

In our demos we load models from google storage buckets. In the model logs we sometimes see this:

```
Compute Engine Metadata server unavailable onattempt
```

This is a known google storage issue but does not cause failures. Treat as a warning.

CHAPTER

TEN

ARCHIVE

10.1 Previous versions

- Version 0.8 Documentation